


PRESENTA

# I miei amici C 16 & Plus 4



I LIBRI DI systems

# I miei amici C 16 & Plus 4



I LIBRI DI
systems

# sommario

Questo libro nasce come sussidio al manuale fornito dalla Commodore agli utenti del nuovo C16 e del Plus 4. Sarà soprattutto di aiuto a coloro che si accingono per la prima volta a "masticare" il BASIC, ma costituirà anche un buon supporto per tutti i possessori di computer Commodore.

Il volume si compone di due parti ben distinte. Nella prima vengono esposte le istruzioni fondamentali del BASIC presente su tutte le apparecchiature della Commodore, nella seconda parte si porrà maggiormente l'attenzione sui potenti comandi del BASIC 3.5 presente sul C16 e Plus 4.

# Approccio ai computer e alla programmazione

Per cominciare a comprendere il funzionamento di un computer, vediamo di costruirne un modello astratto.

Stacchiamoci per un momento dal mondo elettronico per soffermarci su quello umano. Pensiamo ad un impiegato seduto alla sua scrivania, che riceve dal suo principale un determinato compito da eseguire su alcune pratiche che si trovano nell'archivio dell'ufficio. Gli ordini del principale giungono scritti su un foglio di carta. Vediamo di collegare tutto questo al computer.

Immaginiamo che l'impiegato sia il cuore del computer, quella parte che riceve gli ordini e li esegue diligentemente; il foglio di carta del principale è il programma, la serie di istruzioni da eseguire; l'archivio della ditta è la memoria di massa del computer, dove risiedono una grande quantità di informazioni che l'impiegato consulta e modifica cercando ogni volta solo ciò che gli interessa; infine la scrivania può essere considerata la memoria centrale sulla quale troviamo il programma da svolgere e un numero limitato di dati che prima risiedevano nell'archivio e che il computer, per poter lavorare meglio, ha portato in un luogo dove può consultarli più facilmente.

Esistono tre momenti ben distinti che possiamo cosl classificare:

- quello di comunicazione del programma (fase di input);
- quello di esecuzione di ogni singolo ordine (fase di elaborazione);
- il computer consegna a chi gli ha richiesto il lavoro il frutto della sua fatica (fase di output).

Può però accadere che il compito sia complesso e che necessiti di un continuo intervento da parte del richiedente per comunicare non solo all'inizio

ciò che desidera, ma anche per far compiere al computer operazioni diverse a seconda del dato che in quel momento è stato preso in esame.

Di conseguenza può esistere un'alternanza delle fasi di input, esecuzione, output. Inoltre possiamo aver bisogno di operare su più di un gruppo di dati residenti nell'archivio, quindi anche la ricerca di quest'ultimo e il trasferimento sulla scrivania è considerato un input. Allo stesso modo la richiesta del programma può riguardare una modifica nello stesso archivio ed il risultato, cioè l'output può essere la memoria di massa.

Abbandoniamo ora il nostro impiegato e occupiamoci del computer.

Una caratteristica fondamentale è la sua capacità di tenere in memoria la serie di istruzioni necessarie per svolgere qualsiasi elaborazione. Questo insieme di istruzioni, il programma, deve trovarsi nella memoria principale del computer perchè i vari ordini possano essere eseguiti.

Ciascun computer ha un insieme fisso di istruzioni che è in grado di eseguire.

L'unità di controllo (il cuore), richiama le istruzioni dalla memoria una per volta, le decodifica e fa sì che il computer le esegua. Se l'istruzione richiede un'operazione aritmetica, l'unità di controllo trasferisce i dati necessari dalla memoria all'unità aritmetica e logica.

La memoria centrale, l'unità logica e l'unità di controllo costituiscono il nucleo del computer. Insieme sono noti come elaboratore centrale (central processor).

Le apparecchiature periferiche di ingresso (input) e di uscita (output), collegate all'elaboratore centrale, servono per immettere (input) i programmi e i dati nella memoria del computer e per ottenere in uscita (output) i risultati delle elaborazioni.

Le unità di input tipiche leggono e decodificano il significato dei fori perforati sulle classiche schede formato dollaro o su nastro di carta, oppure "sentono" i segnali ottici o magnetici dei supporti appositi e trasmettono elettronicamente queste informazioni all'unità di elaborazione centrale.

# Schema di elaboratore centrale

In alternativa, l'informazione può essere digitata direttamente da una tastiera.

Il risultato può venire visualizzato sullo schermo di un televisore, di un monitor o di altri tipi di display. Se occorrono copie documentali scritte su carta, le stampanti possono riprodurre i caratteri uno per volta, una riga alla volta, su foglio singolo o su carta continua. Per riprodurre dei disegni si usano dei tracciatori grafici speciali (plotter).

I programmi possono essere conservati magneticamente su apparecchiature di memoria di massa per poi, all'occorrenza, rileggerli e trasferirli nella memoria del computer. Nei grandi computer (detti anche mainframe) a questo scopo si usano dischi o nastri magnetici. Nei più piccoli microcomputer si ricorre ai dischetti flessibili (floppy disk) e alle cassette che riescono a contenere meno informazioni e le trasferiscono più lentamente, ma risultano estremamente più economici. Le unità di memoria di massa servono altresì a immagazzinare gli archivi (file) dei dati registrati. Questi, come abbiamo già schematizzato, vengono trasferiti dalla memoria centrale del computer, sotto il controllo del programma che eseguirà ciò che è richiesto.

La figura 1.1 mostra gli elementi costitutivi di base del computer, il flusso dei dati ed i collegamenti di controllo.

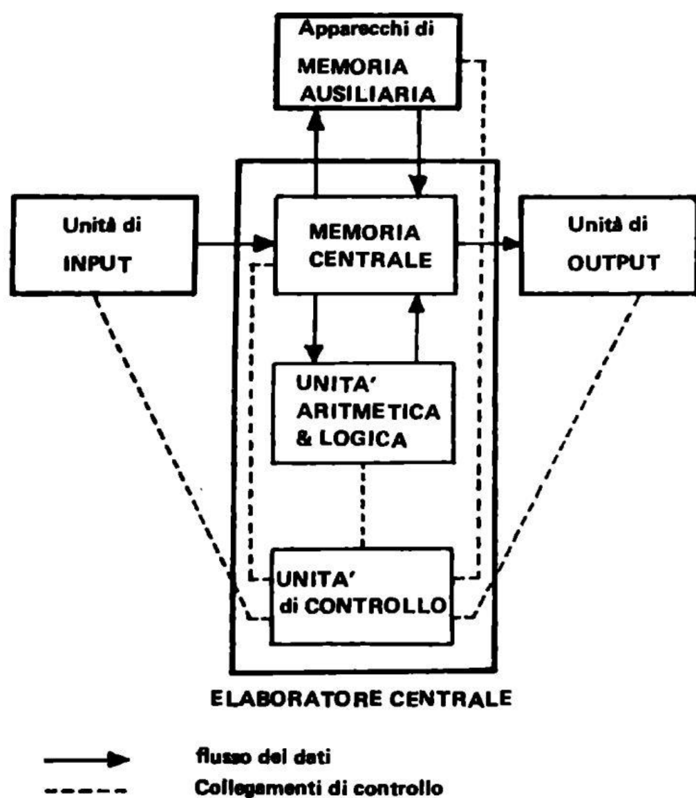



Fig. 1.1 - Elementi costitutivi di un computer

# Gestione delle informazioni

Un computer è costituito da un gran numero di componenti a due stati. Lo stato spento (OFF) convenzionalmente rappresenta uno 0. Lo stato acceso (ON) rappresenta un 1. Un sistema di numerazione che utilizza solo lo 0 e l'1 è detto binario. Combinando opportunatamente queste cifre binarie (dette anche bit binary diggit) si possono ottenere qualsiasi numero e qualsiasi carattere. Nella memoria i numeri sono rappresentati da una combinazione di bit. La quantità di bit utilizzata per rappresentare un numero varia secondo il computer usato.

Con il linguaggio BASIC, la maggior parte dei sistemi lavorano in aritmetica a virgola mobile e i numeri sono memorizzati come mantissa ed esponente. Ad esempio, $6 \times 10^3 = 6.000$ ha come mantissa 6 e come esponente 3.

# La programmazione di un computer

Ogni famiglia di processori ha un proprio insieme di istruzioni, normalmente diverso da quello di tutti gli altri. Ciò significa che ogni dato processore è capace di capire soltanto il suo esclusivo insieme di istruzioni in codice binario. La memoria del computer può essere immaginata come un alveare di celle capaci di immagazzinare le combinazioni binarie rappresentanti le istruzioni del programma e i dati. Ciascuna cella è contraddistinta da un numero esclusivo, grazie al quale è possibile leggere il contenuto o modificarlo scrivendone uno nuovo.

Per dare un esempio di come i programmi sono scritti nel codice proprio di un computer (linguaggio macchina) immaginiamo due numeri memorizzati nelle celle 5 e 6. Questi due numeri possono essere addizionati e il risultato può venire a sua volta conservato, poniamo, nella cella 8. L'addizione richiesta verrà effettuata in una particolare localizzazione di memoria, detta accumulatore (contenuto nel processore stesso). La prima istruzione del programma, pertanto, comanderà di trasferire (in gergo caricare) uno dei due numeri nell'accumulatore. La seconda istruzione richiederà di aggiungere il secondo numero a quello precedentemente scritto nell'accumulatore, nel quale ora si troverà la somma dei due numeri. La terza istruzione mira a trasferire (scaricare) e memorizzare il risultato contenuto nell'accumulatore nella cella di memoria voluta.

La codifica binaria di queste istruzioni, espressa per un processore tipo, è esposta in tabella 1.1. Se nelle celle 5 e 6 avessimo memorizzato rispettivamente i numeri 70 e 25, dopo l'esecuzione delle tre istruzioni, le celle 5 e 6 conterrebbero ancora i numeri 70 e 25, mentre nella cella 8 troveremmo 70 + 25 ossia 95.

Lo stesso programma può essere riutilizzato cambiando i dati delle celle 5 e 6 (per esempio 43 e 12), col risultato che nella cella 8 il precedente valore 95 verrebbe sostituito dal nuovo valore 55.

**Tabella 1.1 - Istruzioni in linguaggio macchina**

| Istruzione | Linguaggio | macchina |
|---|---|---|
| 1 - Carica il numero della cella 5 nell'accumulatore | 10100101 | 0101 |
| 2 - Aggiungi il numero della cella 6 a quello dell'accumulatore | 01100101 | 0110 |
| 3 - Memorizza il numero dell'accumulatore nella cella 8 | 10000101 | 1000 |

# I linguaggi di programmazione

Come abbiamo visto, per programmare in linguaggio macchina, occorre dare le istruzioni e i dati in binario. Ma scrivere e battere sulla tastiera tutti questi 1 e 0 richiede troppo tempo e sicuramente può dare luogo a tanti errori. Per eliminare questi inconvenienti, si ricorre a dei codici mnemonici. Ad esempio, il comando di caricare un numero da una cella di memoria potrebbe essere scritto come LOAD (dall'inglese load t carica) invece che con una serie di bit, tipo 10100101.

Analogamente, alle celle di memoria potremmo attribuire dei nomi simbolici invece di indicarle con i loro veri numeri di indirizzo in binario.

Questo tipo di linguaggio di programmazione aiuta a controllare pienamente le funzioni del computer. I linguaggi che utilizzano simili codici mnemonici sono chiamati linguaggi assemblatori. Ogni istruzione del linguaggio assemblatore normalmente corrisponde ad un'equivalente istruzione in codice macchina. La traduzione di un programma dal linguaggio assemblatore in linguaggio macchina viene eseguita da un particolare programma in codice macchina, detto assemblatore.

Una particolare categoria è quella dei linguaggi ad alto livello, la cui caratteristica prima consiste nel permettere di esprimere con un solo comando tutta una serie di istruzioni in linguaggio macchina. Consentono di astrarsi il più possibile dalle singole istruzioni in linguaggio macchina per avvicinarsi ad un linguaggio di tipo umano. Il BASIC è uno di essi.

Vediamo ad esempio la seguente espressione:

LET C = A + B

dove LET è un termine inglese corrispondente al nostro "sia".

Questa istruzione BASIC fa sì che i contenuti delle celle di memoria A e B siano addizionati e che la somma venga conservata nella cella C. Questo stesso problema in precedenza (tabella 1.1) aveva richiesto tre istruzioni in codice macchina o in linguaggio assemblatore. Ma nè il linguaggio assemblatore nè i programmi in BASIC possono venire compresi direttamente dal computer. A tale scopo questi ultimi debbono essere tradotti in linguaggio macchina mediante un compilatore o interprete.

La differenza consiste nel livello in cui si verifica la traduzione in codice macchina. Con il compilatore essa avviene prima che il programma venga eseguito, cosa che rende molto veloce quest'operazione. L'interprete, pur essendo più lento perchè la traduzione viene effettuata man mano che il programma viene eseguito, è però particolarmente utile per chi si avvicina per la prima volta al computer. E' infatti stato progettato in modo da facilitare al massimo tutte le operazioni di correzione e la modifica dei programmi.

# Microprocessor e microcomputer

Lo sviluppo della microelettronica ha posto il mondo dei computer alla portata di chiunque, mutando altresì il modo in cui queste macchine venivano utilizzate.

Inoltre le ridotte dimensioni e il basso costo dei sistemi basati sul microcomputer rendono possibile l'allargamento dei campi applicativi. I congegni microelettronici (circuiti integrati) sono costituiti da minuscoli e sottili strati di materiale semiconduttore, come il silicio. Un piccolo quadratino (chip, si pronuncia "cip") di silicio della superficie di pochi millimetri quadrati, può contenere un elevatissimo numero di componenti elettronici incorporati nei circuiti. I circuiti integrati (IC) sono disponibili in una vasta gamma in base alle varie funzioni di memoria e di elaborazione.

Oggi è possibile disporre di tutti i circuiti occorrenti per un microcomputer, raggruppati in un unico chip semiconduttore, all'incirca della stessa dimensione dei primi integrati i quali contenevano solo pochi componenti. I circuiti integrati ad elevata integrazione (LSI=large scale integration) contengono diverse migliaia di componenti. In un microcomputer, il chip che svolge le funzioni di processore centrale è chiamato microprocessore. Questo componente svolge in particolare le seguenti funzioni:

- sincronizzazione degli eventi dell'elaborazione e decodifica delle istruzioni (unità di controllo);
- memorizzazione temporanea degli indirizzi e dei dati (registri);
- operazioni aritmetiche e logiche (unità aritmetica e logica).

# Semplici istruzioni di ingresso e uscita: i comandi READ, DATA, INPUT, PRINT

In questo capitolo vedremo i comandi che regolano l'immissione dei dati nel computer e il modo di visualizzare sullo schermo i risultati delle elaborazioni.

Ciascuna istruzione BASIC dà l'ordine al computer di svolgere una data funzione, ognuna con una propria "sintassi" che regola il modo con cui vengono scritti i singoli comandi e tutto ciò che completa l'istruzione.
Poniamo il caso di dover chiedere al computer di leggere dei valori (numerici) che invieremo dalla tastiera e di memorizzarli in alcune porzioni di memoria identificate ognuna da un carattere, per esempio A, che indica una parte della memoria che può contenere un numero reale (per esempio 125879,12).

Per compiere questa operazione, dovremo innanzitutto ordinare al computer di leggere, in seguito dovremo dirgli dove andare a leggere e che cosa. In BASIC questa sequenza può essere scritta così:

```
INPUT A
```

In questo modo ordiniamo al computer di leggere il numero che noi batteremo e di memorizzarlo in un gruppo di celle di memoria del C 16 e del PLUS 4, identificate dalla lettera A. Questa lettera, viene chiamata variabile; è qualcosa di simile a un cassetto della scrivania contraddistinto da A, nel quale diciamo al computer di mettere (o prendere) ciò che ci occorre. Quindi, durante tutta l'esecuzione di un programma, quando parleremo di A, il computer saprà che ci riferiamo a quel cassetto.

In BASIC i differenti "cassetti" che possono contenere numeri reali vengono sempre indicati con lettere dell'alfabeto, dalla A alla Z, eventualmente seguite da un numero (da 0 a 9) o da un'altra lettera.

Ora cominciamo a scrivere il nostro primo programma, usando appunto l'istruzione INPUT. Prima digitiamolo e poi esamineremo singolarmente quanto abbiamo scritto.

```
10 INPUT A
20 PRINT A
30 END
```

Battete questo programma separando le linee con il tasto RETURN che riporterà il cursore a capo, pronto per scrivere di nuovo. Come vedete, ogni linea è preceduta da un numero che indica al computer l'ordine con cui deve eseguire le istruzioni. In questo modo possiamo modificare il nostro programma qualora volessimo inserire un'istruzione tra due già scritte.

Per poter compiere quest'operazione è però necessario, in fase di scrittura del programma, numerare le linee con multipli di 10 (come è stato fatto sopra).

Così, se ora digitiamo:

```
15 INPUT B
25 PRINT B
```

avremo inserito due nuove linee nel nostro programma.

Come avete visto, abbiamo battuto un'istruzione per linea. Ciò non sarebbe necessario perchè è possibile scrivere due istruzioni sulla stessa linea separandole con un ":"; in questo modo però la lettura ed la comprensione risultano più agevoli.

Rivediamo ora il nostro programma, digitando LIST seguito da RETURN. Il computer ci riproporrà tutto il programma, comprese le correzioni inserite dove necessario.

Se abbiamo compiuto tutte le operazioni correttamente, vedremo comparire sullo schermo:

```
10 INPUT A
15 INPUT B
20 PRINT A
25 PRINT B
30 END
```

Dopo aver eseguito il comando LIST, il computer scriverà:

READY

Questo segnale indica che il computer è pronto per eseguire un nuovo comando (in inglese ready vuol dire pronto).
Ora digitiamo RUN, sempre seguito da RETURN e chiediamo di eseguire il programma. Il computer risponderà scrivendo un punto di domanda. Ciò significa che sta eseguendo l'istruzione INPUT e sta attendendo che vengano scritti dei numeri, seguiti sempre dal tasto RETURN, che serve, oltre a portare il cursore a capo, anche a "inviargli" ciò che abbiamo scritto, sia esso un comando, una linea di programma o, come in questo caso, un INPUT (quando in seguito parleremo di scrivere un programma o di far eseguire un nuovo comando, sarà sempre necessario usare il tasto RETURN come si è fatto sino ad ora).

Dopo aver dato al computer il primo numero, ci verrà riproposto un secondo punto di domanda per ricevere il secondo numero. Quando l'avremo scritto e memorizzato nella variabile assegnata, verrà eseguita l'istruzione successiva, PRINT, che permette di visualizzare sullo schermo i due numeri battuti e tutto ciò che vogliamo: contenuti di variabili numeriche (come nel nostro caso), scritte di segnalazione ecc...

Vediamo come sfruttare quest'istruzione modificando il nostro programma. Ribattiamo LIST per richiamare su schermo il testo. Ora possiamo cancellare una linea scrivendo il suo numero e battendo subito RETURN; in questo modo possiamo, per esempio, eliminare la linea 25. Adesso possiamo posizionarci con il cursore sulla linea 20 dopo la lettera A, qui aggiungiamo i caratteri ",B" sempre seguiti da RETURN. Digitando LIST 20, ci verrà visualizzata la linea 20 che ora sarà così:

```
20 PRINT A,B
```

Se rieseguirete il programma tramite RUN, vedrete che i numeri non saranno più stampati su due righe differenti, ma appariranno sulla stessa linea ad una certa distanza. Possiamo ancora effettuare una modifica sostituendo la virgola tra A e B con un punto e virgola.

```
20 PRINT A;B
```

Anche così i due numeri saranno sulla stessa linea, ma sarà diversa la loro distanza. Ci accorgiamo quindi che possiamo ottenere risultati differenti dalla stessa istruzione, a seconda di come scriviamo e separiamo i suoi vari campi.

Vediamo ora come utilizzare ancora l'istruzione PRINT per scrivere dei messaggi. Possiamo digitare:

```
5 PRINT"SCRIVI DUE NUMERI
```

Così sarà più chiaro, per chiunque usi il programma, cosa deve scrivere quando compare il punto di domanda.

Abbiamo visto come si possono inserire dei messaggi. Ma poichè molto spesso si richiede un input da tastiera ed è facile che, se questo non è preceduto da un commento, chi usa il programma non sappia cosa debba scrivere, esiste la possibilità di fondere le due istruzioni in una sola.

Cambiando la 10 così:

```
10  INPUT"SCRIVI IL PRIMO NUMERO";A
20  INPUT"SCRIVI IL SECONDO";B
```

potremo eliminare la linea 5 oramai superflua.

Cominciamo a conoscere due nuove istruzioni molto semplici. Cancelliamo il programma con il comando:

```
NEW
```

In questo modo abbiamo "pulito" la memoria del computer e quindi non esistono più nè il programma nè le variabili A e B.

Ora possiamo scrivere un nuovo programma:

```
10  READ A,B,C
20  PRINT A,B,C
30  DATA 15,123,658
40  END
```

Questo programma usa una sola istruzione che già conosciamo e contiene due nuovi comandi: READ e DATA che, anche in futuro, vedremo sempre comparire abbinati tra loro.

READ in inglese significa "leggi"; noi abbiamo già visto un'istruzione che fa qualche cosa di simile: la INPUT. La differenza tra le due è che una legge da tastiera e l'altra dall'istruzione DATA.

Nel caso del nostro programmino, READ assegnerà alle variabili A,B,C rispettivamente i valori scritti dopo DATA, cioè 15, 123, 658. Questo modo di assegnare dei valori alle variabili è meno elastico di quello ottenuto con INPUT. Infatti i valori DATA vengono determinati durante la scrittura del programma invece che durante l' esecuzione e ciò può essere utile quando dobbiamo usare un valore costante invece di uno variabile.

# Variabili stringa

Per molti problemi occorre immettere, memorizzare ed ottenere in output delle informazioni variabili consistenti in un misto di lettere, numeri e caratteri speciali inclusi gli spazi. Queste serie di simboli sono dette *stringhe.*

Le stringhe possono essere memorizzate in variabili stringa. Ad esse occorre dare un nome consistente in una lettera dell'alfabeto (eventualmente seguita da un'altra lettera o da un numero da 0 a 9,) accompagnata dal segno di dollaro $, come ,ad esempio, in A$, B$, C$,...Z1$. Le informazioni costanti date fra doppie virgolette sono dette costanti stringa.

Le variabili stringa sono essenziali per leggere e lavorare con gli archivi e soprattutto nelle applicazioni gestionali.

Vediamo alcuni esempi sul loro uso.

Dopo che un programma è stato scritto ed è stata accertata la sua correttezza, può essere usato all'infinito con dati diversi, in diverse occasioni e da diverse persone. E' consigliabile, tuttavia, prendere nota delle date in cui il programma è stato utilizzato e da chi. Per immettere questa informazione (e poi leggerla) possono essere usate due variabili stringa.

Scriviamo ora un programma che farà parte di quello in cui si vuole memorizzare la data di utilizzo.

```
10 IMPUT"SCRIVI LA DATA DI OGGI";N$
20 PRINT
30 PRINT "LA DATA DI OGGI E'";N$
40 END
```

La linea venti serve solo per stampare una linea bianca per rendere più elegante e leggibile l'output.

# Stampare un testo

E'importante progettare degli output adeguati, in modo che si adattino a formati e agli obiettivi più diversi.

Per illustrare ciò, vedremo i vari sistemi per l'uso delle informazioni anagrafiche (nome e indirizzo).

Il programma di tabella 2.1 consente di immettere un titolo (Sig., Sig.ra ecc.) un nome e un indirizzo, memorizzandoli in variabili stringa e di ottenere la stampa d'una intestazione di lettera.

A seguito delle sei istruzioni INPUT di tabella 2.1, il computer vi richiederà d immettere sei righe di dati, ciascuna in risposta al ? che lampeggia sul video (vedi tabella 2.2).

Scrivete questo programma, battendolo come avete imparato in precedenza, poi provate a farlo eseguire; se tutto funziona, memorizzatelo su cassetta battendo SAVE "TABELLA 2.1", seguito da RETURN. Il computer vi dirà tutto quello che dovete fare.

**Tabella 2.1**

```
10 INPUT T$
20 INPUT N$
30 INPUT A$
40 INPUT B$
50 INPUT C$RECT
60 INPUT D$
61 PRINT"●":REM CANCELLA SCHERMO
70 REM INTESTAZIONE LETTERA
80 PRINT  TAB(3);A$
90 PRINT  TAB(3);B$
100 PRINT TAB(3);C$
110 PRINT TAB(3);D$
120 PRINT
130 PRINT
140 END
150 REM ETICHETTA PER AGENDA
160 PRINT
170 PRINT
190 PRINT"***********"200 PRINT
210 PRINT
220 PRINT TAB(5);N$
230 PRINT
240 PRINT
250 PRINT"***********"260 PRINT
270 PRINT
280 END
290 REM ETICHETTA PER BUSTA
300 PRINT
310 PRINT
320 PRINT
330 PRINT TAB(5);T$;" ";N$
340 PRINT TAB(5)A$
350 PRINT TAB(5)B$
360 PRINT TAB(5)C$
370 PRINT TAB(5)D$
380 END
```

Lo dichiarazioni REM (da remarks t note, commenti) vengono inserite nel programma allo scopo di spiegare le attività del programma nei diversi punti. Il computer sa che questi commenti non sono istruzioni da eseguire.

Il comando "PULIZIA DELLO SCHERMO", mostrato tra virgolette nella riga 61 è ottenuto premendo contemporaneamente i tasti SHIFT e CLR/HOME. Facendo ciò, cancellerete dallo schermo tutto ciò che avete scritto in procedenza, in modo che l'intestazione della lettera venga visualizzata in alto a sinistra sullo schermo.

A questo punto, troverete una nuova istruzione che si combina a PRINT: TAB(n), dove nè il numero che indica quanti spazi vuoti devono precedere la stampa della variabile o della stringa.

```
SIG.
? MARIO ROSSI
? CORSO SEMPIONE 95
? 20149 MILANO
?
?
```

# Le operazioni aritmetiche

## Costanti e variabili

Opportunamente programmato, il computer può effettuare una grand varietà di calcoli, dal più semplice al più complesso, avendo inoltre la possibil tà di assegnare il risultato ad una variabile. Ad esempio:

```
LET S=X+Y
```

dice al computer di sommare il contenuto della variabile X a quello della Y e conservare il risultato nella variabile S. Naturalmente i valori di X e Y sono sta definiti in precedenza mediante un'istruzione INPUT o il tandem di istruzio READ + DATA o, come vedremo ora, mediante un'altro LET. L'effetto di LE non modifica il contenuto delle celle X e Y.

Ad esempio, la tabella 3.1 mostra il contenuto delle celle di X, Y e S prim dell'esecuzione di questo programma:

```
10 READ X, Y
20 DATA 123,56
30 LET S=X+Y
```

**Tabella 3.1 - Contenuto di X,Y ed S**

| Cella | Dopo | Prima |
|---|---|---|
| X | 123 | 123 |
| Y | 56 | 56 |
| S | 179 | ? |

Notate come sull'originario contenuto di S (sconosciuto) è stato scritto il nuovo valore 179, risultante dal valore 123 + 56. I fattori che troviamo nella parte destra del segno uguale di una istruzione LET possono sia dipendere da diversi operatori aritmetici che essere costituiti da valori costanti.

Un esempio:

```
51 LET I=I -1
```

sottrae 1 dall'attuale valore di I, sicchè, dopo l'esecuzione dell'istruzione LET, I assume un nuovo valore inferiore di 1 rispetto al precedente.

Va notato che la parola LET può essere omessa.

Le classi numeriche utilizzabili sono:

- numeri interi (come 1, 12, 789, ecc...) che non contengono valori decimali, cioè i numeri con la virgola (gli angloamericani e anche i computer usano il punto al posto della nostra virgola decimale. Per gli inglesi la virgola serve a separare le migliaia invece del nostro punto);
- numeri reali, in forma decimale quando il numero da rappresentare è relativamente piccolo, in forma di potenze del 10 (mantissa più esponente) se il numero è grande (per grande si intende lontano dallo zero, quindi sia positivo che negativo.

Ricordiamoci che per fare assumere a un numero un valore, è sufficiente premettergli il segno (-); il segno (+) o l'assenza di un segno specifico indica che il numero è positivo.

Qualsiasi numero utilizzato in un programma, sia che si tratti di una costante che del contenuto di una variabile, deve essere contenuto nei limiti di capacità del computer e cioè:

+-(1.70141183)*E↑38, +-(2.93873588)*E↑- 39

rispettivamente per il massimo ed il minimo numero.

## Operatori aritmetici

I simboli posti alla destra del segno = d'una istruzione LET possono essere nomi di variabili, costanti ed operatori aritmetici. Tale combinazione di simboli è detta espressione aritmetica.

Gli operatori aritmetici indicano quali operazioni vanno effettuate sui numeri dell'espressione aritmetica. Il seguente elenco mostra in quale ordine vengono svolte le varie operazioni, a meno che la loro sequenza non venga modificata nell'uso di parentesi, come vedremo in seguito.

| *Operatori aritmetici* | *Significato* |
|---|---|
| ↑ | elaborazione a potenza (esponenziazione) |
| *, / | moltiplicazione, divisione |
| +, - | somma, sottrazione |

# Gerarchia delle operazioni

L'uso delle parentesi serve a modificare la sequenza e la priorità delle operazioni da svolgere. Il contenuto delle parentesi va svolto a cominciare dalla coppia posta più all'interno, procedendo verso quelle più esterne.

Ad esempio per calcolare:

$$\frac{5+9}{4+3}$$

è necessario calcolare prima la parte superiore (numeratore), quindi quella inferiore (denominatore) e infine dividere il numeratore per il denominatore. Per indicare questa sequenza di operazioni si ricorre alla parentesi.

Un programma che spiega questa sequenza di calcolo è illustrato in tabella 3.2.

Lo spazio maggiore lasciato nella numerazione delle istruzioni LET e PRINT consente il successivo inserimento di altre istruzioni di cui potremmo avere bisogno.

**Tabella 3.2 - Programma per illustrare l'ordine di calcolo**

```
30 READ B,C,D,E
40 DATA 5,9,4,3
50 LET A=(B+C)/(D+E)
70 PRINT B;C;D;E;A
90 END
```

**Tabella 3.3**

| Variabile | B | C | D | E | A |
|---|---|---|---|---|---|
| Prima | 5 | 9 | 4 | 3 | ? |
| Dopo | 5 | 9 | 4 | 3 | 2 |

La tabella 3.3 illustra il contenuto delle variabili prima e dopo l'esecuzione dell'istruzione LET della linea 50. Fate eseguire questo programma sul vostro computer, quindi modificate l'istruzione LET come segue (ossia eliminate le parentesi:

50 LET A=B+C/D+E

Otterrete 10.25 (cioè 9/4+5+3). Ciò perché senza le parentesi il vostro computer svolgerà i vari calcoli secondo un suo ordine. In particolare eseguirà, per prima cosa, gli elevamenti a potenza, quindi le moltiplicazioni e/o le divisioni nell'ordine in cui sono scritte, infine le addizioni e/o le sottrazioni procedendo sempre verso destra.

Se osservate ancora l'ultima istruzione alla linea 50, noterete che la divisione, C/D, è stata eseguita prima delle addizioni. Il risultato dunque, sarà notevolmente diverso da quello calcolato con la precedente istruzione LET di tabella 3.2.

## Espressioni aritmetiche e istruzioni

Inserite nel programma le istruzioni LET e PRINT illustrate in tabella 3.4 e fatelo eseguire.

**Tabella 3.4 - Istruzioni LET**

```
50 LET A=(B+C)/(D+E)
51 LET G=C/E-B*D
52 LET H=C/(E-B)*D
53 LET J=G-H/E+E↑2
61 LET S=C*D-B↑A
62 LET T=(C*D-B)↑A
63 LET U=(E*(C-B)↑(D/A))
70 PRINT
71 PRINT
72 PRINT"B=";B;"C=";C;"D=";D;"E=";E
73 PRINT
74 PRINT
75 PRINT"A=";A;"G=";G;"H=";H;"J=";J
76 PRINT
77 PRINT
78 PRINT"S=";S;"T=";T;"U=";U
80 END
```

Visualizzate tutti i risultati (A,G,H,J,S,T,U) e i nomi delle variabili a cui si riferiscono. I dati letti ed i risultati finali sono mostrati in tabella 3.5

**Tabella 3.5 - Dati letti e risultati finali**

| | | | |
|---|---|---|---|
| B = 5 | C = 9 | D = 4 | E = 3 |
| A = 2 | G = -17 | H = -18 | J = -2 |
| S = 11 | T = 961 | U = 48 | |

Come ultimo esercizio modificate le espressioni aritmetiche delle linee 51-63 del listato 3.4 sostituendole con quelle di tabella 3.6. Confrontate i risultati con quelli di tabella 3.7. Il computer vi dà la possibilità di cambiare i singoli caratteri di ogni istruzione; non cambiate quindi tutta la linea.

**Tabella 3.6 - Espressioni aritmetiche modificate**

```
51 LET G=C/(E-B)*D
52 LET H=C/(E-B*D)
53 LET J=((G-H)/E+E)↑2
61 LET S=C*(D-B)↑A
62 LET T=C*(D-B↑A)
63 LET U=E*C-B↑D/A
```

**Tabella 3.7 - Risultati di espressioni aritmetiche**

| | | | |
|---|---|---|---|
| B = 5 | C = 9 | D = 4 | E = 3 |
| A = 2 | G = -18 | H = -.529412 | J = 7.97232 |
| S = 9 | T = -189 | U = 285.5 | |

# Lo sviluppo dei programmi

Questo capitolo insegna a sviluppare i programmi di cui avete bisogno.

Se un programma è scritto troppo frettolosamente, perderete molto tempo nella correzione e nei cambiamenti che in seguito si renderanno necessari. Le ore impiegate nella preparazione dei vostri programmi non sono mai sprecate. Quanti progettano sistemi gestionali o ne scrivono i programmi devono conformarsi ad una serie di procedure formali. Analogamente, anche chi sviluppa programmi per proprio conto deve abituarsi all'osservanza di certe regole.

## Progettazione dell'output

Il primo passo nella progettazione di un programma è lo studio dell'output. In particolare dovete riflettere e prendere decisioni su alcuni punti fondamentali.

Come sapete, il risultato di un programma può essere visualizzato, stampato su carta e/o trasferito in archivio. Per questo dovete decidere con esattezza quale scelta operare.

Ad esempio, il vostro obiettivo è scrivere un programma per leggere l'archivio dei dati di un magazzino e produrre un elenco degli articoli da riordinare. Per ottenere un simile elenco dovete prendere alcune decisioni preliminari:

- l'output richiesto deve consistere soltanto in un elenco dei prodotti da riordinare, o deve essere al contempo l'input per un programma per l'emmissione degli ordini d'acquisto?
- Nel primo caso, quali dati dovete stampare?

• E' necessario stampare tutti i dati dell'archivio relativi ai vari articoli da ordinare o vi bastano solo i codici di questi articoli?

Un programma di questo tipo sarà esposto successivamente. L'elenco, in questo caso, comprende sia i codici che le descrizioni degli articoli da riordinare. Non è stato stampato il contenuto dell'intero record, ma solo i dati necessari all'esatta identificazione di ciascun articolo.

Dopo d'aver deciso quale sarà il contenuto dell'output, bisògna riflettere sul formato e sull'impostazione generale. Ecco le principali considerazioni da fare:

- in quali colonne vanno stampate le variabili?
- Le cifre debbono essere troncate o arrotondate?
- Occorre inserire dei titoletti per ogni colonna?
- Quanti spazi occorrono tra i titoletti e le colonne?
- E' opportuno sottolineare i titoletti?

**Tabella 4.1 - Titoletti per una distinta di riordino**

```
10 PRINT"PRINT"
20 PRINT"STAMPA"
40 PRINT"LISTA  DI RIORDINO"
50 PRINT
60 PRINT"CODICE";TAB(7);"DESCRIZIONE"
65 PRINT"------";TAB(7);"-----------"
70 PRINT
```

L'output d'un programma per l'emissione di una distinta di riordino potrebbe cominciare come mostrato in tabella 4.1.

## Dati di input occorrenti

Dopo aver deciso le caratteristiche dell'output, occorre identificare i dati di input necessari. Se i dati da elaborare sono numerosi, è consigliabile leggerli da un apposito archivio (sempre che esista o sia possibile crearlo); di quest'argomento parleremo successivamente.

I dati occorrenti solo per il programma specifico possono essere immessi mediante dichiarazioni DATA, mentre è preferibile inserire tramite istruzioni INPUT quelli che cambiano tutte le volte che si fa girare il programma.

Poichè è possibile che voi non siate l'unico utilizzatore di un programma, i vari valori debbono essere immessi nella loro forma più comune, ad esempio dando 12.5 invece di 125 per un tasso di interesse.

La visualizzazione di una serie di esaurienti messaggi aiuteranno l'utente ad utilizzare il programma con maggiore facilità guidandolo, ad esempio, nella fase di immissione dei dati.

Un ulteriore aspetto della progettazione dell'input è rappresentato dal desiderio di disporre di qualche forma di controllo del programma durante la sua esecuzione.

Ad esempio, nel problema del "riscaldamento" all'utente viene chiesto se desidera elaborare ulteriori dati. L'utilizzatore interagisce rispondendo S o N:

```
100 PRINT "ALTRI DATI?"
105 PRINT "(S = SI,N=NO)"
110 INPUT YS$
```

## Diagrammi di flusso

Quando le vostre idee saranno abbastanza chiare, cominciate a sviluppare la sequenza logica delle istruzioni del programma. Questa fase può essere realizzata disegnando appositi diagrammi di flusso (in inglese flow-chart).

I simboli più comuni utilizzati a questo scopo sono illustrati in figura 4.1; un esempio del loro uso è dato in figura 4.2, che illustra il processo per il calcolo della media aritmetica fra tre numeri. L'obiettivo di un diagramma di flusso è quello di garantire che la logica sia corretta, prima di passare alla scrittura più dettagliata delle singole istruzioni. Nel corso di questo stesso volume vedremo ulteriori esempi di diagrammi di flusso a illustrazione di altri programmi.

Spesso il diagramma o l'analisi del problema evidenziano che una data operazione, nel corso dello stesso programma, verrà ripetuta numerose volte. Quando è evidente che una stessa sequenza di istruzioni potrebbe essere riutilizzata, val la pena scriverla una sola volta come sottoprogramma in modo da poterlo utilizzare quando serve. In seguito vedremo dettagliatamente queste subroutine.

Il passo successivo alla stesura del diagramma di flusso è la scrittura vera e propria del programma. Ma il lavoro non finisce qui, una fase importantissima consiste nella sua prova, al fine di accertarne la piena rispondenza agli effetti desiderati.

Vediamo dunque come provare e documentare i programmi

| *Simbolo* | *Uso* |
| --- | --- |
| (terminale) | Inizio e fine dell'elaborazione |
| (rettangolo) | Fase dell'elaborazione |
| (rombo) | Prova condizionale che conduce a tre sequenze di elaborazione alternative |
| (parallelogramma) | Dichiarazione di input o di output |
| (cerchio A) | Connettore (consente al diagramma di allacciarsi ad un punto di connessione A). |

**Figura 4.1 - Alcuni simboli dei diagrammi di flusso**

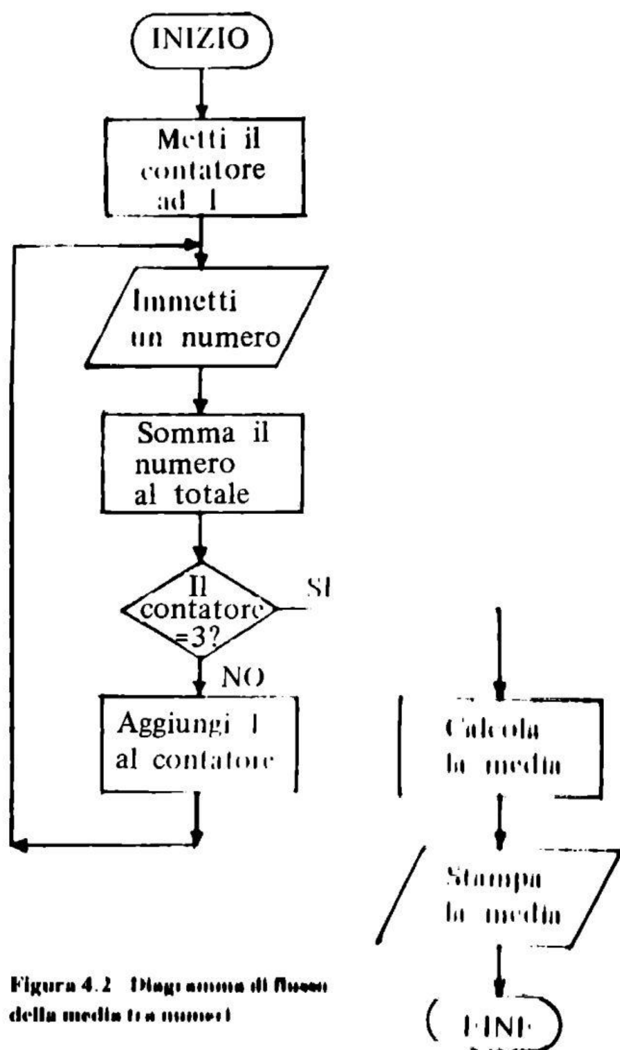



**Figura 4.2 Diagramma di flusso della media tra numeri**

## La prova dei programmi

Se nello scrivere programmi BASIC commettete un errore, il computer lo rileverà in fase di esecuzione e invierà un messaggio per informarvi che il vostro programma contiene un *syntax error* (errore di sintassi o istruzione sbagliata).

Esempi tipici sono gli errori di digitazione (es. IPUT invece di INPUT), la formulazione sbagliata delle istruzioni (es. LET X + Y = S invece di LET S = X +Y), nomi di variabili inaccettabili (es.*A invece di PA ). Prima di andare avanti dovremo chiarire e mettere a punto tutti i syntax error.

Ma anche dopo questa ripulitura il vostro programma può non essere ancora del tutto corretto. In particolare potreste essere incorsi in un errore di esecuzione (*execution error*) chiedendo al computer qualcosa che invece non può eseguire.

Ad esempio, se i valori calcolati dal vostro programma sono troppo grandi o troppo piccoli, causereste un overflow (uscireste dal campo in cui il calcolatore sa operare), cosa che succede anche se cercate di dividere un numero per zero.

Un altro errore di esecuzione di verifica se tentate di assegnare una stringa a una variabile numerica (ad esempio usando D invece di D$ per la data 07/02/85).

Un programma che funzioni con successo, senza che errori di esecuzione, potrebbe non essere ancora a punto, a causa della sua logica non corretta. Pertanto, prima di eseguirlo, dovrete provarlo sulla carta con dati di cui conoscete già il risultato. Questa operazione viene chiamata elaborazione a tavolino.

Successivamente, eseguite il programma sul computer con gli stessi dati, scelti in modo da testare tutte le sequenze di istruzioni.

E' importante mettere per iscritto tutti i dettagli del programma e il modo di utilizzarlo, per consultarli in un secondo tempo. Affinchè la vostra documentazione possa essere completa, strutturatela seguendo questo schema:

a - titolo di identificazione;
b - indice del contenuto;
c - riassunto;
d - descrizione del problema;
e - specifiche del problema;
f - moduli per l'output;
g - uso del programma;
h - interpretazione dell'output;
i - modifiche;
l - appendici eventuali.

# Salti condizionati e incondizionati

## Il controllo dei livelli di priorità delle istruzioni

La maggior parte dei problemi richiede che il computer sia programmato in modo da ripetere una o più serie di istruzioni secondo le esigenze del momento. Ciò è possibile per mezzo delle istruzioni di salto (jump).

Una di esse è GOTO (vai a), che determina il passaggio dell'esecuzione al numero di linea indicato. In altre parole, il computer esegue l'istruzione verso la quale è "saltato" e poi tutte le altre che seguono in sequenza, finchè non incontra un'altra istruzione di salto.

Ad esempio:

```
50 LET I=1
60 PRINT I
70 LET I=I+2
80 GOTO 60
```

determina la stampa dei numeri dispari 1,3,5 ecc. Tutte le volte che il computer trova l'istruzione alla linea 80, salta automaticamente alla 60 eseguendo l'istruzione che vi si trova. Passa, poi, all'istruzione 70 e via di seguito.

Possiamo quindi dire che GOTO è un'istruzione di salto incondizionato, dato che viene eseguito indipendentemente dalle condizioni esistenti.

Nel pezzo di programma sopra riportato non esiste nessuna istruzione che fermi il programma. In BASIC, la possibilità di far compiere un salto condizionato è fornito dall'istruzione IF... THEN (se... allora). Per interrompere, ad esempio, il programma che stampa numeri dispari, potete aggiungere al gruppo di istruzioni riportato più sopra, le due che seguono:

```
65 IF I=21 THEN GOTO 90
90 END
```

Eseguite quindi il programma, verificando se effettivamente si interrompe dopo la stampa del numero 21. Se sostituite il 21 con un numero pari, diciamo 20 o 22, il programma non si interromperà mai, dal momento che I non assumerà mai questo valore.

# I loop e il loro controllo

Il programmino che abbiamo appena esaminato è costituito da un set di istruzioni, dalla linea 60 alla 80, strutturate in forma ciclica, di loop (loop significa anello, cerchio).

Per avere un'idea chiara di ciò che si intende per loop e per salto fuori dal loop, sarà sufficiente dare un'occhiata allo schema a blocchi riportato in figura 5.1. Noterete che l'istruzione GOTO 60 è rappresentata da una freccia che va dal riquadro 70 al riquadro 60. I modi per uscire da un loop e saltare verso parti diverse di un programma sono numerosi. Osservate il formato dell'istruzione IF...THEN:

*numero di linea IF espressione relazione THEN salto a una linea con numero diverso.*

Il numero di linea che segue la parola THEN deve essere diverso da quello che precede la parola IF, altrimenti è l'istruzione IF stessa a causare il loop.

L'espressione relazionale costituisce la verifica da eseguire, il test che determina l'istruzione seguente. Se è vera (cioè, se la condizione sussiste), il controllo passa al GOTO successivo a THEN; se è falsa passa al numero della linea successiva a IF e le istruzioni continuano a essere eseguite in sequenza finchè non viene raggiunta un'altra istruzione di salto. L'espressione relazionale confronta due espressioni nel formato seguente:

espressione operatore relazionale espressione

Se ricordate, avete già usato un operatore relazionale nel capitolo precedente: si tratta dell'operatore =.

Per avere comunque una lista completa di tutti gli operatori relazionali disponibili, confrontate la tabella seguente:

**Tabella 5.1 - Gli operatori relazionali**

| operatore relazionale | significato |
|---|---|
| = | uguale a |
| > | maggiore di |
| < | minore di |
| >=*o*=> | maggiore di o uguale a |
| <=*o*=< | minore di o uguale a |
| <> *o*>< | non uguale a |

L'istruzione IF...THEN è particolarmente utile per controllare l'introduzione dei dati; può, infatti, essere usata per verificare la correttezza dei risultati finali, individuando la presenza di un eventuale valore particolare o di un valore indicante la fine dell'elenco dei dati, che non verrà usato nell'esecuzione di calcoli nell'ambito del programma.

Il concetto è illustrato in tabella 5.2, che contiene un programma di somma di numeri positivi, in cui i numeri vengono introdotti uno per volta in risposta all'istruzione INPUT della linea 30.

**Tabella 5.2 - Una chiusura di programma con un valore falso**

```
10 PRINT"SOMMA DI N NUMERI"
11 PRINT
20 LET T=0
30 INPUT X
40 IF X<0 THEN 70
50 LET T=T+X
60 GOTO 30
70 PRINT "TOTALE =";T
80 END
```

Il programma di tabella 5.2 si fermerà quando in X verrà letto o uno zero o un valore negativo. L'istruzione IF...THEN deve comparire prima dei calcoli che coinvolgono la X, in modo che il valore che determina la conclusiaone del loop non alteri il risultato. Un altro modo per interrompere l'esecuzione di un set di istruzioni consiste nello specificare quante volte il loop deve essere eseguito, come indicato in tabella 5.3.

**Tabella 5.3 - Programma che esegue la somma di N numeri**

```
10 INPUT N
20 PRINT"SOMMA DI";N;"NUMERI"
25 PRINT
30 LET I=0
35 LET I=0
40 INPUT X
50 LET T=T+X
60 LET I=I+1
70 IF I<N THEN 40
80 PRINT"TOTALE =";T
90 END
```

Se la linea 30 di tabella 5.3 avesse letto:

30 LET = 1

la linea 70 avrebbe dovuto essere:

70 IF I < = N THEN 40

Questo perchè quando il loop è stato eseguito N volte, se I è fissato in modo da partire da 1, il valore di I, dopo che è stata eseguita la linea 60 è maggiore di uno rispetto al numero dei numeri di linea. Ciò significa che il loop termina non appena I=N + 1.

## Confronto fra stringhe di carattere

Poichè ogni carattere memorizzato nel computer è rappresentato da un'unica combinazione di cifre binarie, è possibile usare l'istruzione IF...THEN per effettuare il confronto anche fra stringhe di caratteri, per esempio, per verificare se P$ contiene il carattere H.

L'istruzione:

25 IF P$ = "H" THEN 30

(l'istruzione GOTO può essere sottintesa, questa è una comoda facilitazione che il BASIC del C16, del Plus 4 e di tutte le macchine Commodore concede) significa che, se P$ contiene H, la condizione è vera e quindi il salto alla linea 30 può essere effettuato.

Questa possibilità è particolarmente utile per confrontare fra loro nomi, indirizzi e altre informazioni di questo tipo, usate specialmente nelle applicazioni gestionali.

Per scoprire quali caratteri hanno un valore superiore o inferiore per i test "maggiore di" o "minore di", avrete bisogno di potervi riferire a un elenco di codici usati per rappresentare i caratteri nella memoria del computer.

## Le istruzioni FOR... NEXT

Nei paragrafi precedenti il numero di volte in cui il loop doveva essere eseguito veniva programmato impostando un valore iniziale per il contatore di loop, sottoponendo a opportuno test il valore finale e incrementando il valore corrente del contatore di loop se quello finale non era ancora stato raggiunto.

L'istruzione FOR...NEXT (per... prossimo), ha lo scopo di semplificare la programmazione di queste tre operazioni.

Nell'esempio della somma di N numeri in tabella 5.3, la variabile I (usata come contatore di loop) è stata impostata sul valore iniziale 0, diventato poi 1, dopo la lettura del numero e la sua somma. Terminata questa operazione, è stato eseguito un test IF I<N, per stabilire se il programma doveva ritornare all'inizio del loop o doveva fermarsi.

Con l'istruzione FOR...NEXT il programma può subire delle interessanti modifiche. Ogni istruzione FOR...NEXT è costituita da due linee di codice. All'inizio del loop, stabilisce le condizioni iniziali, l'incremento o passo (step) da eseguire alla fine del loop e il valore finale. Possiamo sintetizzare il ciclo di FOR in questo modo:

*numero di linea FOR variabile t espressione 1 TO espressione 2 STEP espression e 3.*

L'espressione 1 stabilisce il valore iniziale del contatore del ciclo (noto come indice), l'espressione 2 quello finale e l'espressione 3 l'incremento da sommare alla variabile alla fine di ogni esecuzione del set di istruzione del loop. Se STEP è uguale a 1, sia la parola STEP che l'espressione 3 possono essere omesse.

L'istruzione finale del loop si presenta nel formato seguente:

*numero di linea NEXT variabile*

La variabile ha lo stesso nome di quello dato nella relativa istruzione FOR.

**Tabella 5.4 - Programma di somma alternativo**

```
10 INPUT N
20 PRINT"SOMMA DI";N;"NUMERI"
25 PRINT
30 LET T=0
35 FOR I=1 TO N
40 INPUT X
50 LET T=T+X
60 NEXT I
80 PRINT"TOTALE =";T
90 END
```

Il programma riportato in tabella 5.3 può essere migliorato nel modo indicato in tabella 5.4. In X viene letto un numero N volte, in base a quanto stabilito dalle istruzioni FOR .... NEXT.

Nella linea 35, I viene posto inizialmente a 1, quindi viene incrementata di 1 nella linea 60 e, se è maggiore di N, esce dal ciclo e prosegue dalla linea 80, che stampa il totale; altrimenti ritorna alla linea 40.

Inserite l'istruzione

70 PRINT I

in modo che, dopo che il loop è stato eseguito per il numero di volte stabilito, possiate vedere il valore di I. Esso può anche essere usato dentro il ciclo, facendo però attenzione a non modificarne il valore, dato che l'operazione produce una variazione delle condizioni stabilite dall'istruzione FOR...NEXT.

Il problema sintetizzato nel primo esempio di questo capitolo può essere codificato come segue:

```
50 FOR I=1 TO 21 STEP 2
60 PRINT I
70 NEXT I
80 END
```

Il valore dell'incremento dato nell'espressione successiva a STEP può essere negativo (decrementanto così il contatore) o frazionario.

La tabella 5.5 riporta un programma esemplificativo in cui è possibile introdurre come variabili i valori di inizio, di chiusura e di incremento del ciclo, cioè A, B e C. Fate delle prove con un certo numero di combinazioni diverse (compresi i valori negativi e quelli frazionari) e osservate quello che succede.

**Tabella 5.5 - Variabili di inizio, di chiusura e di incremento**

```
5 PRINT"DIGITA L'INIZIO,"
10 PRINT"LA FINE E"
15 PRINT"PASSO DESIDERATO"
16 PRINT
20 INPUT A,B,C
30 FOR I=A TO B STEP C
40 PRINT I
50 NEXT I
60 END
```

## L'istruzione ON...GOTO

L'istruzione ON...GOTO (nel caso che...vai a) si presenta nella forma seguente:
*numero di linea ON espressione GOTO due o più numeri di linea separati da virgole*

La parte intera dell'espressione valutata dev'essere un numero positivo non superiore alla quantità di numeri di linea che vengono dopo la parte GOTO dell'istruzione.

Il controllo passerà al primo, secondo, terzo... numero di linea dopo GOTO solo se la parte intera dell'espressione sarà uguale a 1, 2, 3...

Una parte di programma che usa il comando ON...GOTO può essere di questo tipo:

```
.
20  IMPUT "DECISIONE 1,2,3?";X
30  ON X GOTO 30,40,50
```

A seconda del valore che noi assegneremo a X, se 1, 2 o 3, il programma continuerà rispettivamente dalla linea 30, 40, o 50.

## Impiego della funzione TAB e dei cicli FOR

La funzione TAB può essere usata con le variabili o con le espressioni, chiuse fra parentesi e poste dopo la parola TAB; ad esempio:

TAB (I), TAB (P-1)

Il programma riportato in tabella 5.8 genera un rettangolo di dimensioni variabili costituito da due lati orizzontali tratteggiati (L1) e da due linee verticali formate dalla I maiuscole (L2).

**Tabella 5.8 - Programma per la generazione di rettangoli**

```
20 PRINT"POSIZ.DELLA COLONNA INIZIALE"
30 INPUT P
40 PRINT"LARGHEZZA E ALTEZZA"
50 INPUT L1,L2
60 LET K=1
```

```
70 PRINT TAB(P-1)
80 REM LINEE ORIZZONTALI
90 FOR I=1 TO L1
100 PRINT"-";
110 NEXT I
120 PRINT
130 IF K=2 THEN 240
140 REM LINEE VERTICALI
150 FOR I=1 TO L2-2
160 PRINT TAB(P-1);"↑";
170 FOR J=1 TO L1-2
180 PRINT" ";
190 NEXT J
200 PRINT"↑"
210 NEXT I
220 LET K=K+1
230 GOTO 70
240 END
```

Le linee 20 e 40 generano un messaggio per l'utente al quale viene chiesto di introdurre dei dati.

L'istruzione PRINT della linea 70 è chiusa da un punto e virgola, che fa in modo che la successiva istruzione PRINT avvenga sulla stessa linea.

La linea 120 serve a generare la linea tratteggiata. Dopo l'esecuzione della linea 220, che stabilisce che K è uguale a 2, vengono ripetute, fino al completamento del rettangolo, le linee da 90 a 120, dopo di che terminerà l'esecuzione del programma.

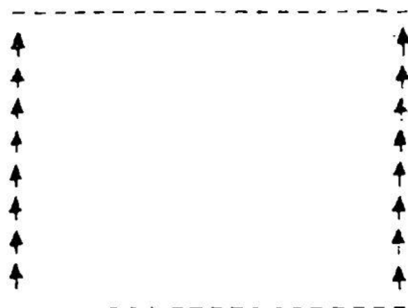
Le linee dal 150 al 210 contengono un ciclo FOR che, a sua volta, ne contiene un altro (linee 170-190).

Questo tipo di loop FOR è detto "annidato" e ne vedremo successivamente le caratteristiche. Ad ogni passaggio attraverso il loop FOR esterno, quello interno viene eseguito LI-2 volte, in modo che venga generata una I seguita da un certo numero di spazi e da un'altra I. Non appena è raggiunta la linea 190, viene eseguito un altro passaggio attraverso il ciclo esterno, finchè non sono state generate L2-2 linee costituite come descritto in precedenza.

In questo caso, l'uso dei cicli FOR annidati potrebbe essere evitato sostituendo il contenuto delle linee da 160 a 220 con l'istruzione seguente:

```
160 PRINT TAB(P-1);"I";TAB(L1-1+P);"I"
```

**Figura 5.2 - Rettangolo generato da programma**



La figura 5.2 mostra un rettangolo generato dal programma precedente ed eseguito con questi dati:

7
20 , 10

i quali stanno a indicare che sono stati generati 20 trattini per le due linee orizzontali e 8 I per le due verticali.

Vi consigliamo, per acquistare una certa pratica, di eseguire il programma riportato in tabella 5.8 introducendo dati diversi.

Usando poi i caratteri grafici per i quattro angoli e i quattro lati, potrete scrivere un altro programma che generi un rettangolo di larghezza ed altezza variabili.

# Manipolazioni di stringhe

## Codici di carattere

Nel computer i caratteri e i numeri vengono memorizzati sotto forma di combinazioni di cifre binarie. I codici binari standard sono stati fissati da diverse organizzazioni internazionali, ma quelli più largamente adottati sono i codici definiti dalla American Standard Code for Information Intercange (ASCII).

A questo punto è necessario aprire una piccola parentesi.

Abbiamo già accennato che la memoria centrale del computer è composta da tante piccole celle, in ognuna delle quali sono contenute otto cifre binarie. Il codice ASCII adopera, per rappresentare tutti i caratteri presenti in ogni computer, sette di questi bit. Il numero massimo di caratteri che possiamo ottenere dalla combinazione di sette cifre binarie è 128 (da 0 a 127 compresi). L'ordine è il seguente:

- da 0 a 31 troviamo i caratteri di controllo quali, ad esempio, RETURN che è il 13;
- da 32 a 47 troviamo gli operatori matematici, oltre a caratteri con significati particolari quali: $, #, eccetera;
- da 48 a 57 vengono rappresentate le cifre da 0 a 9 in ordine crescente;
- da 65 a 90 ci sono le lettere maiuscole;
- da 97 a 122 troviamo i caratteri minuscoli.

Nel Commodore C16 e nel Plus 4, per lasciare maggior spazio ai caratteri grafici, viene rispettato il codice ASCII solo fino alle maiuscole; per ottenere le minuscole viene usato un diverso artifizio che per ora non ci interessa. Per

curiosità, se volete vedere diventare minuscoli tutti i caratteri, premete il tasto SHIFT insieme al tasto Commodore (l'ultimo in basso a sinistra). I caratteri possono essere convertiti in questi codici, e viceversa, attraverso le funzioni BASIC ASC e CHR$, che ora descriveremo.

• *La funzione CHR$*. Dà il carattere corrispondente ad uno specifico codice ASCII, cioè:

```
10 LET A$ = CHR$ (66)
```

Il codice ASCII per la lettera B è 66, quindi l'istruzione memorizza B in A$. Le parole vengono formate mediante concatenazione, vale a dire:

```
10 LET A$ = CHR$(66) + CHR$ (69)
```

dà per risultato:

A$ = BE

dove 69 è il valore ASCII corrispondente alla lettera E. Notate che poichè la funzione CHR$ dà il codice ASCII, le variabili possono essere usate, se necessario, per diversi caratteri di controllo (per esempio il RETURN).

• *La funzione ASC*. E' l'opposto della funzione CHR$, in quanto fornisce il numero di codice ASCII per un dato carattere.

Ad esempio:

```
10 LET X = ASC("E")
```

dà come risultato:
X = 69

Se l'argomento è una variabile stringa, viene fornito il codice ASCII del primo carattere; ad esempio:

```
 5 LET T$ = "TOTAL"
10 LET X = SC(T$)
```

danno per risultato:

X = 84

che è il valore ASCII del carattere T.

• *LEN.* Questa funzione fornisce la lunghezza di una stringa. Cambiando, ad esempio, la linea 10 in modo da avere:

10 LET L = LEN T (T$)

si avrebbe come risultato:
L = 5.

• *Le funzioni LEFT$ e RIGHT$.* Individuano il numero specifico di caratteri più a sinistra o più a destra di una data stringa. Così ad esempio:

10 LET B$ = LEFTS$(TS,2)

fornisce i due caratteri più a sinistra della stringa T$, cioè:

B$ = TO

e

10 LET E$ = RIGHT$(T$,3)

dà per risultato:
E$ = TAL

• *La funzione MID$.* Estrae una sottostringa a partire dal carattere che si trova nella posizione indicata dal numero che segue il nome della stringa; il numero successivo indica invece quanti caratteri dovranno essere presi, ad esempio:
10 LET C$ = MID$ (T$,2,3,)

dà per risultato:
C$ ="OTA"

dove 2 indica il secondo carattere dal quale deve cominciare e 3 quanti caratteri dopo il secondo verranno presi.

• *La funzione STR$.* Converte un argomento numerico in una stringa contenente come caratteri le cifre che compongono il numero. Per esempio:

```
10 LET N = 16
20 LET X$ = STR$ (N)
```

dà per risultato:
X$

contenente "16" come stringa, perciò:

```
30 LET Y$ ="COMMODORE C" + X$
40 PRINT Y$
```

dà come risultato la stampa dell'espressione:

COMMODORE C16

• *La funzione VAL*. E' l'esatto contrario della funzione STR$: essa infatti esamina la stringa. Se il primo carattere che la compone è una cifra, prosegue prendendo tutte le altre immediatamente successive e, mantenendone l'esatto ordine, compone il numero corrispondente; nel caso in cui il primo carattere non fosse numerico restituisce come valore 0.

Vediamo un esempio, continuando il programma precedente:

```
50 LET V = VAL(Y$)
60 PRINT V
```

darà come risultato:
0

Se invece scriviamo un nuovo programma:

```
10 LET A$ = "10COMMODORE C16"
20 LET V = VAL (A$)
30 PRINT V
```

darà come risultato:
10

Perchè il carattere che segue lo zero non è una cifra, ignora quindi la presenza del successivo 16.

# Altre funzioni

## Funzioni matematiche

Il BASIC, inizialmente nato come linguaggio per chi comincia ad avvicinarsi alla programmazione, fu intelligentemente fornito di primitive (istruzioni) per l'esecuzione di parecchie funzioni matematiche che, in precedenza, erano appannaggio solo di alcuni linguaggi specialistici.

Nel BASIC sono presenti routine usate di frequente, come quelle necessarie per ottenere la parte intera (INT), il logaritmo e l'antilogaritmo di un numero (LOG ed EXP) e le funzioni trigonometriche (ad esempio SIN); è possibile inoltre definire funzioni matematiche per ogni necessità. Vediamo dettagliatamente quali sono.

## Argomenti

Tutti i nomi di funzione sono seguiti da un'espressione (argomento) fra parentesi. La funzione viene calcolata sull'argomento. La sintassi è per quasi tutti questa:

funzione "(" argomento ")"

L'argomento può essere, a seconda dei casi, una costante, una variabile, un risultato di una operazione tra variabile e costanti o ancora un'altra funzione. Sono infatti anche previste funzioni di funzioni, come del resto succede molto frequentemente in campo scientifico-matematico.

Per esempio:

```
100 LET S = SQR(B↑2 - 4*A*C)
```

calcola la radice quadrata dell'espressione fra parentesi (che rappresenta il determinante delle equazioni di secondo grado): B al quadrato - 4 per A per C e lo assegna alla variabile S.

Per ciò che riguarda il valore dell'argomento relativo ad una certa funzione, ci possono essere limitazioni dettate da certe regole, le stesse, ovviamente, utilizzate in matematica: per esempio non è possibile estrarre la radice quadrata di un numero negativo, ecc...

## La funzione INT

Abbiamo già accennato alla funzione INT, che permette di ottenere la parte interna di un numero, vale a dire un numero intero minore dell'argomento della sua parte decimale. Se quest'ultimo è un numero positivo, le cifre decimali vengono semplicemente eliminate.

L'istruzione:

```
110 LET B = INT (A)
```

se A è uguale a 15,36 pone in B il solo valore 15.

Ricordate però che dopo l'esecuzione dell'istruzione LET, A non viene modificato e continua a contenere il valore con le cifre decimali. Se invece A contiene il valore - 15,36 il valore intero posto in B non è - 15 (dato che è maggiore di - 15,36), ma - 16.

Se volessimo ottenere un valore intero maggiore di un numero negativo, prima di usare la funzione INT, occorrerà innanzitutto togliere il segno usando la funzione ABS, che considera il valore assoluto del suo argomento (il valore assoluto di un numero è il numero stesso se è maggiore di zero, o l'opposto del numero se esso è minore di zero) e poi utilizzare la funzione SGN (signum). Quest'ultima fornisce il valore 1 se il suo argomento ha un valore positivo, - 1 se il suo argomento ha un valore negativo e 0 se il valore del suo argomento è zero.

Così ad esempio:

```
50 LET B = SGN(A)*INT(ABS(A))
```

In questa assegnazione si moltiplica la parte intera del valore assoluto di A per il suo segno, in modo da avere la parte intera di A, minore di A, se esso è positivo, maggiore di A se è negativo.

Ad esempio:

```
10 LET A = -123.234
.
60 PRINT B
```

darà come risultato -123, mentre:

```
10 LET A = 324.891
```

darà come risultato 324.

## Abbreviare e arrotondare

Spesso, prima ancora di compiere operazioni su certi dati, si è già a conoscenza del fatto che i risultati non saranno completamente esatti, ma che esisteranno piccoli valori, in più o in meno rispetto al valore esatto, indotti proprio dal tipo di calcolo applicato oltre che dai dati presi in considerazione.

Vediamo un piccolo esempio.

Pensiamo di elaborare dati contenenti parecchie cifre decimali. Possiamo avere quindi numeri del tipo:

```
10 LET A=0.9999999999
```

Esso differisce dall'unità (1) di un piccolissimo valore: 1 E-10, vale a dire:
0.0000000001

Proviamo ora ad effettuare questa assegnazione:

```
20 LET B =A+A
```

Se facessimo il conto a mano otterremmo il valore:
1.9999999998

che è diverso da 2 a meno di 2 E↑10. Provate a vedere cosa invece calcola il computer:

```
30 PRINT B
RUN
```

Siete stupiti vero? In B infatti avete trovato il valore 2. L'errore, come abbiamo visto, è molto piccolo, dell'ordine di 10 alla - 10, ma se a noi fosse interessato solo la parte intera del numero, ora avremmo un errore dell'ordine di una unità!

Per non avere brutte sorprese è decidere quanto dovrà essere grande l'errore, così da non illuderci di aver ottenuto risultati esatti quando questi, purtroppo, non lo sono. Se infatti, con il programmino di prima, partivamo dal presupposto che del nostro dato A consideravamo "significative" (sicuramente esatte) tutte le cifre decimali, ci saremmo aspettati un risultato che ci restituisse più o meno la stessa precisione; invece neppure la prima cifra intera poteva essere considerata significativa. Completiamo il programmino così:

**Tabella 7.1 - Arrotondamento al primo più prossimo**

```
20 PRINT"ANGOLO   GRADI MINUTI"
30 PRINT
40 INPUT A
50 IF A=0 THEN 110
60 D=INT(A)
70 REM ARRORONDAMENTO
80 M=INT((A-D)*60+0.5)
90 PRINT A;TAB(10);D;TAB(17);M
100 GOTO40
110 END
```

Qui viene introdotto un altro modo di operare. Il primo, quello appena visto, non ci permette di sapere quante cifre decimali possiamo considerare significative. Il secondo, che abbrevia il numero troncandolo alla terza cifra decimale, per poi approssimarlo per difetto di 5·E↑3, ci consente di sapere che il risultato non sarà quello esatto, ma che sarà compreso tra C + o - 5·E ↑3.

C-5E-3 < x <C+5·E-3

Ovviamente, se noi modifichiamo la linea 40 aggiungendo .5 invece di sottrarlo, ci allontaneremo maggiormente dal risultato esatto, ma saremo sempre all'interno dell'intervallo sopra definito.

Volendo poi diminuire l'intervallo di indeterminazione, possiamo aumentare il numero degli zeri presenti nella linea 40 così:

```
40 LET C = INT(A/.00001+.5)*.00001
```

## Radici quadrate

Il calcolo delle radici quadrate viene eseguito dalla funzione standard SQR. Anch'essa obbedisce alla regola sintattica sopra indicata. Quando si introduce in un programma questa funzione, bisogna ricordare che essa non può operare su tutto il campo reale, in quanto non è possibile, nel campo dei numeri reali, estrarre la radice di un numero negativo. E' quindi buona norma eseguire un test con la funzione SGN per controllare se è possibile compiere l'operazione. Un esempio estrapolato da un programma potrebbe essere:

```
.
20 INPUT X
.
100 IF SGN(X) < 0 THEN PRINT"ERRORE": GOTO 20
110 LET S = SQR(X)
```

## Funzioni trigonometriche

Insieme a quelle già viste, il BASIC è anche fornito di tutte le funzioni necessarie per compiere calcoli trigonometrici, ampiamente usati in matematica e sopratutto in fisica. Possiamo infatti ottenere i valori del seno, coseno, tangente e arcotangente di angoli la cui misura deve essere espressa in radianti.

Le funzioni sono rispettivamente:

SIN, COS, TAN, ATN

che naturalmente rispettano la sintassi:

funzione (argomento)

In molti interpreti BASIC, per facilitare ulteriormente i calcoli trigonometrici, possiamo trovare già disponibile come costante il valore "pi greco"; in quelli residenti nelle macchine della famiglia Commodore ciò non avviene, ma è possibile ottenerlo con la formula:

```
10 LET PI = ATN(1)*4
```

con l' accortezza di non fare, durante tutto il programma, ulteriori assegnazioni alla variabile PI.

## Logaritmi e antilogaritmi

Sempre sotto forma di funzione BASIC possiamo avere la possibiltà di calcolare il logaritmo o l'antilogaritmo di una espressione. Le primitive BASIC sono rispettivamente le funzioni LOG e EXP. La funzione LOG permette di calcolare il logaritmo naturale di un'espressione, cioè il logaritmo in base "e" (numero neperiano) dal quale è possibile ottenere il valore del logaritmo in base dieci attuando la conversione secondo la nota regola:

$$\frac{\log_e x}{\log_e 10} = \log_{10} x$$

La funzione EXP dà invece il valore della costante matematica "e", numero di Nepero (2.7182...) elevato alla potenza del suo argomento; essa è legata al logaritmo naturale dalla formula:

$$\log_e N = x \text{ se } e^x = N$$

Il numero di Nepero, lo precisiamo per chi userà il BASIC non solo per operare in campo scientifico matematico, ma anche apprendere le nozioni di queste scienze proprio attraverso il computer, è il limite per x tendente a infinito della funzione:

$$(1 + 1/x)^x$$

## Funzioni definite dall'utente

Esiste anche la possibilità di costruire personalmente una funzione alla quale consegnare, di volta in volta, valori differenti della variabile ad essa associata. E' necessario quindi prima definire la funzione e poi assegnare i differenti valori della variabile a questa riferita.

La modalità per definire una funzione è, ad esempio, la seguente:

```
10 DEF FN A(X)= X↑2-2*X+5
```

In questo caso il nome della funzione è FN A.

Quando, all'interno del programma, desideriamo effettuare il calcolo della funzione per il valore di X=10, scriveremo:

```
90 FN A(10)
```

Così, implicitamente assegnamo alla X della funzione FN A il valore 10, il calcolatore andrà a cercarsi la funzione corrispondente al nome ed effettuerà il calcolo rendendo così il valore richiesto.

Tutte le funzioni definite in questo modo devono avere all'interno del programma un solo nome definito dalla DEF FNx(y), dove x può essere chiamato identificatore della funzione ed y è la variabile sulla quale vengono compiuti i calcoli.

Come per tutte le altre funzioni, anche per quelle definite dall'utente è possibile costruire delle funzioni di funzioni.

Per esempio:

```
10 DEF FN S(X)=X↑2 -4
20 DEF FN A(Y)=ABS(FN S)*Y↑2 -6/Y
```

Come si può vedere, il valore della seconda funzione dipende, oltre che da Y, anche dal valore della FN S

In questo capitolo abbiamo visto la quasi totalità delle funzioni presenti nelle librerie di sistema della maggior parte degli interpreti BASIC su macchine Commodore e non solo su queste.

Ne esistono altre, ma data la loro stretta somiglianza con quelle analizzate fino ad ora, possiamo in coscienza lasciare a voi il compito di scoprirle in base alle vostre necessità e al computer su cui lavorate. Per quanto riguarda C16 e Plus 4, rimandiamo alla seconda parte del testo.

# Le subroutine

Finora abbiamo visto piccoli programmi che eseguivano, a meno di qualche salto condizionato (IF... seguito da GOTO), in sequenza tutte le istruzioni scelte.

Spesso, però, all'interno di uno stesso programma è necessario far eseguire, in momenti diversi, una specifica serie di istruzioni. Sarebbe poco intelligente riscriverle tutte le volte dove ci occorrono perchè sprecheremmo spazio di memoria, sempre molto prezioso.

Occorre quindi scriverle una sola volta e far si che il programma salti, all'occorrenza, a questo gruppo di istruzioni, che chiameremo subroutine, cioè sottoprogramma (una sorta di programma più piccolo contenuto in uno generale).

La subroutine, dunque, può essere "chiamata" ogniqualvolta occorra e possiede inoltre una prerogativa molto importante: dopo la sua esecuzione il programma continuerà dalle istruzioni che seguono la chiamata stessa. E' quindi necessario, oltre a un salto per la chiamata, anche un salto per il ritorno.

Per ricordarci ogni volta qual è la linea di programma successiva alla chiamata, dovremmo giostrare con delle variabili usate come segnalatori e, con l'istruzione ON...GOTO, decidere a che punto di programma ritornare. La cosa però può diventare scomoda, specialmente se le chiamate diventano parecchie; sarebbe più utile, quindi, se fosse il calcolatore stesso a "ricordarsi" l'istruzione seguente la chiamata.

E' proprio ciò che fa la coppia di istruzioni "GOSUB numero di linea", RETURN. Con GOSUB chiamiamo la subroutine, mentre con RETURN ritorniamo al punto successivo la chiamata. Esiste uno spazio di memoria dedicato alla memorizzazione dei valori che indicano il punto successivo la chiamata, si chiama *stack* (pila) ed è organizzato appunto come una pila di tipo LIFO (il primo ad essere immesso è l'ultimo che uscirà) nella quale è possibile scrivere questi valori quando viene eseguito GOSUB, per poi riprenderli, togliendoli dalla pila, quando si esegue RETURN.

La struttura a pila LIFO si rende necessaria qualora dall'interno di una subroutine volessimo effettuare un'altra chiamata a subroutine; dovremmo infatti memorizzare anche questa posizione e tornare qui prima di restituire l'esecuzione al GOSUB della prima chiamata.

Fortunatamente tutte queste operazioni sono eseguite dal computer, ma il discorso è stato fatto proprio per capire che in un programma dovrà esistere un numero di subroutine limitato alla dimensione del nostro stack; attenzione quindi a non abusarne, anche perchè, come vedremo tra poco, anche i cicli FOR...NEXT usano questa pila.

# Vettori e matrici

Finora, nei nostri piccoli programmi, abbiamo sempre adoperato variabili di tipo semplice, ad ogni nome era associato un valore o una stringa di caratteri. Esiste invece una seconda categoria di variabili un po' più complessa, ma molto utile.

Vediamo di giungere a un'immagine figurata di ciò che sono le matrici e i vettori. Tutti voi, almeno una volta nella vita, avete giocato a battaglia navale. Vi ricorderete che il terreno di gioco era composto da tante caselline, ognuna contraddistinta da due valori: le coordinate riga e colonna che si scrivevano sui bordi del foglio. Per raggiungere ogni casella dovevamo quindi conoscere più di un identificatore. Le matrici sono molto simili a questo modello.

Le matrici più semplici sono i vettori. Essi sono come una fila sola del foglio da battaglia navale; abbiamo un solo identificatore che determina ogni casella, perchè l'altro è unico (0) e viene sottinteso. Le matrici invece possono avere più numeri per identificare le caselle. Nella battaglia navale le dimensioni erano due: righe e colonne, ma possiamo anche immaginare matrici con tre dimensioni come i tre assi cartesiani che definiscono lo spazio tridimensionale. Ogni casella contiene una variabile. Per esempio:

A$(6)

identifica una variabile stringa che occupa la settima posizione (esiste anche A$(0)) del vettore A$(n), dove n è il numero massimo (meno uno) di caselle del vettore A$.

Possiamo tranquillamente usare vettori che contengano fino alla casella nove, dopo la quale è necessario comunicare al computer la dimensione della matrice perchè esso possa riservare lo spazio di memoria sufficiente.

L'istruzione per dimensionare i vettori e le matrici è:

10 DIM A$(100,10)

Con questa istruzione abbiamo dimensionato una matrice a due dimensioni che è composta da ben 1111 variabili (101*11). Osserviamo un piccolo esempio applicativo:

```
10 INPUT"DIMENSIONE VETTORE";A
20 LET A1=A-1
30 DIM A$(A1)
40 FOR I=0 TO A1
50 INPUT"NOME";A$(I)
60 NEXT I
80 FOR I=0 TO A1
90 PRINT A$(I)
100NEXT I
200END
```

Vediamo quindi come è facile manipolare vettori, ma anche matrici con i loop di FOR.
Infatti, se invece di una sola dimensione ne avessimo avute due, per esempio:

```
30 DIM A$(N,K)
40 FOR I=0 TO N
40 FOR J=0 TO K
50 INPUT A$(I,J)
60 NEXT J
70 NEXT I
```

avremmo dovuto "annidare" un FOR nell'altro per poter scandagliare per ogni valore di N tutti i valori di K.

Come per le subroutine, anche gli indirizzi di ritorno al FOR dopo ogni NEXT nei loop vengono salvati sullo stesso stack di cui si parlava, per cui è bene, anche ai fini della velocità di esecuzione, non esagerare nell'annidare tanti cicli, soprattutto dopo molte chiamate consecutive a subroutine quando ancora non è stato chiuso il primo RETURN.

Per quanto riguarda i FOR annidati, vi raccomandiamo di non eseguire programmi che escono da questi loop prima che vengano terminati; ciò può causare errori difficili da interpretare e correggere. E' quindi buona abitudine rispettare questa norma. Ad esempio, se noi alle linee precedenti aggiungessimo:

```
50 IF A$(I,J)="@" THEN GOTO 1230
```

non daremmo la possibilità al computer di "deallocare". cioè togliere dallo stack gli indirizzi di ritorno dei due FOR.

# Uso dei file

Finora abbiamo visto come possiamo memorizzare i dati solo nella memoria centrale del computer e, anche se conosciamo strutture di variabili come le matrici, ci troviamo di fronte ad un grosso problema. Sappiamo, infatti, che i dati immessi nel calcolatore, una volta spento, vengono inevitabilmente persi. Non possiamo permetterci di mantenere acceso il computer ininterrottamente per tutto il tempo che ci occorrono certi dati; oltretutto non potremmo nel frattempo usare la macchina per effettuare altri lavori.

Come abbiamo visto per i nostri programmi, esiste la possibilità di "salvarli" su supporti magnetici, nastro o disco, per poterli richiamare nella memoria centrale ogni volta che ci occorrono. E' possibile compiere queste operazioni anche con i dati contenuti nelle variabili, organizzate in strutture chiamate *file*. Esiste quasi sempre una struttura, di cui è composto ogni file, che raggruppa tutte le variabili da registrare in "pacchetti" contenenti sempre lo stesso tipo di variabili: il *record*. L'esempio più classico è l' archivio di un magazzino, vediamone un esempio molto semplice ma esplicativo.

Immaginiamo che questo magazzino abbia un certo numero di prodotti dei quali ci interessa sapere in ogni momento la quantità di merce presente. Ogni prodotto è identificato da un numero di codice. Possiamo avere quindi per ogni prodotto questa successione di dati:

- numero di codice;
- nome del prodotto;
- quantitativo giacente in magazzino;
- commenti vari.

Ognuno di questi campi può essere associato a una variabile, per esempio:

N contiene il numero di codice;
N$ contiene il nome;
Q contiene il quantitativo;
V servirà per commenti vari.

L'insieme di queste variabili è detto appunto record (blocco da registrare). nostro file sarà quindi composto da una lunga sequenza di n record dell stesso tipo. Il programma dovrà quindi essere in grado di generare la struttur del record, costruire un file formato da quei record, registrare e successiva mente rileggere ogni record che ci interessa. Esistono diversi modi di risolver il problema ed esistono diversi tipi di file. Noi parleremo solo dei file sequen ziali perchè sono gli unici che possono essere registrati anche su nastro.

# Alcune considerazioni

Con i file si può dire concluso il discorso riguardante il BASIC presente su tutti i computer della famiglia Commodore. Non tutti gli argomenti sono stati approfonditi allo stesso modo, ma pensiamo che quanto scritto sia più che sufficiente per cominciare a "pestare" sulla vostra tastiera e per entrare nel grande mondo della programmazione. Come riassunto vi presentiamo un programma, in sè molto semplice, che si propone l'obiettivo di riassumere quasi tutte le istruzioni fin qui trattate.

E' necessaria però una piccola spiegazione di questo programma. L'algoritmo da cui è stato preso appartiene alla famiglia dei cosiddetti sort. Questi tipi di algoritmi studiano le varie tecniche per "ordinare" degli elementi, nel nostro caso delle stringhe di un vettore. Il sort più semplice in BASIC si chiama Bubble Sort, ordinamento a bolle. Il suo nome deriva dalla similitudine tra le bollicine di una bibita gasata che salgono verso l'alto e gli elementi che via via "salgono" nel vettore per occupare la posizione ordinata.

Lasciamo a voi lo spunto per personalizzare il programma; volendo, potreste in seguito scaricare su nastro il vettore ordinato, o anche accoppiare altri campi a ogni nome, quali numeri di telefono e indirizzi.

Vi ricordiamo che, al crescere del numero di elementi, oltre al disordine con cui sono disposti, aumenta il tempo di esecuzione. Non allarmatevi, se avete 100 nomi dovrete aspettare anche qualche minuto prima che il computer abbia finito.

# Il BASIC 3.5 del C 16 e Plus 4

E' tempo di affrontare anche quelle istruzioni BASIC che esulano un poco da ciò che abbiamo visto finora e che permettono al BASIC di difendersi onorevolmente nella lotta tra i vari linguaggi di programmazione.

Per evitare di travolgervi con una marea di nuove istruzioni, abbiamo scelto di approfondirne una selezione: quelle che possono essere particolarmente utili per l'apprendimento delle tecniche di programmazione. In appendice presenteremo poi tutti i restanti comandi e istruzioni che aumentano considerevolmente le possibilità, soprattutto dal punto di vista grafico. Approfondiremo le istruzioni e i comandi che facilitano la lettura e scrittura dei programmi. oltre che quelli che introducono alla programmazione strutturata

# I comandi AUTO e RENUMBER

Ormai avete già scritto abbastanza programmi da esservi un po' stancati di digitare tutte le volte il numero di linea. In effetti, esiste un comando che permette la scrittura automatica di questi numeri con l'incremento che si vuole. Il comando ha una sintassi molto semplice:

AUTO numero di incremento

Tutte le volte che terminate una riga con il tasto RETURN, il computer vi proporrà il nuovo numero di linea lasciando il cursore nella posizione esatta per cominciare a scrivere la nuova linea. Per uscire dalla modalità auto, basta scrivere AUTO senza nessun numero.

Spesso però, sarà necessario aggiungere delle linee al programma, che inevitabilmente non potrà più seguire l'ordine di numerazione precedente. Il comando RENUMBER permette di rinumerare parte o tutto il programma in questo modo:

• RENUMBER seguito da tre valori opzionali che assegnano rispettivamente il numero di linea nuovo, il valore di incremento, il numero di linea da cui partire; i valori sono separati da virgole. Per esempio:

RENUMBER 100,20,15

rinumera il programma dalla linea 15 in poi assegnando ad essa il valore 100, crescendo poi di 20 in 20;

• omettendo i primi due parametri, verranno rinumerate di 10 in 10 le linee a partire dalla 15:

RENUMBER ,,15

• senza nessun parametro, rinumererà tutto il programma con multipli di 10:

RENUMBER,,.

# I comandi OPEN e CLOSE

Prima di iniziare a registrare dati in un file, dobbiamo segnalare al computer la nostra intenzione. Si usa il comando OPEN che possiede questa sintassi:
OPEN x,y,z,"nome file"
dove x,y,z sono parametri che hanno questo significato:
x = numero di identificazione del file; viene associato un numero a ogni file aperto con il quale esso verrà richiamato all'interno del programma:
y = numero del device (periferica) che conterrà il file;
z = numero che determina le possibili operazioni che si compiranno sul file (lettura o scrittura) nel caso del nastro o il numero del canale nel caso del disco. Per il nastro il numero 0 apre il file in lettura, l'1 in scrittura, il 2 in scrittura con il marcatore di fine nastro.

Per scrivere o leggere nei file usiamo gli stessi comandi che già conosciamo: PRINT per scrivere e INPUT per leggere. Essi devono però essere seguiti dal carattere "#".

Quando sono terminate le operazioni di lettura o scrittura, è necessario chiudere il file con il comando CLOSE seguito dal numero del file.

Proviamo a scrivere un primo programma (tabella 9.1).

**Tabella 9.1 - Record di magazzino**

| N. | Descrizione | Magazzino | Costo unitario | Quantitativo nuovo ordine |
|---|---|---|---|---|
| 1234 | Penne | 15 | 45 | 20 |
| 2340 | Matite | 50 | 12 | 40 |
| 2679 | Gomme | 8 | 5 | 10 |
| 3456 | Righelli | 20 | 26 | 30 |
| 4567 | Blocchetti | 40 | 35 | 50 |
| 4568 | Blocchi | 60 | 40 | 30 |
| 6770 | Etichette adesive | 70 | 15 | 25 |
| 6775 | Puntine da disegno | 40 | 15 | 20 |
| 6979 | Buste | 40 | 20 | 60 |
| 7050 | Libri di cassa | 30 | 22 | 40 |

In esso compaiono quasi tutte le strutture viste finora, tranne le subroutine; lasciamo a voi la scelta di rendere più "elegante" questo programma strutturandolo a subroutine.

Noterete che per terminare la scrittura dei dati si usa la tecnica del "dandy record", o record stupido, il quale non contiene dati che occorrono, ma serve a determinare la fine della serie di record sia in fase di output su nastro che, soprattutto, in fase di input.

# Altri comandi: HELP, RESUME, TRON, TROFF

Quando, durante l'esecuzione di un programma, l'interprete incappa in un errore, viene interrotta l'esecuzione e viene stampato un messaggio d'errore contenente il tipo di errore verificatosi e il numero di linea in cui si trova. Se non ci ricordiamo l'esatto contenuto della linea, il comando HELP ci permette di visualizzare la sola linea in cui ha trovato l'errore.

Per vedere se anche nella rimanente parte di programma è stato commesso un errore, con il comando RESUME possiamo mandare in esecuzione il programma dalla linea d'errore, dalla seguente, o da una qualsiasi;

- RESUME senza parametri riesegue il programma dalla linea d'errore;
- RESUME NEXT esegue il programma dalla linea seguente l'errore;
- RESUME seguito dal numero di linea esegue il programma a partire da quella linea senza cancellare le variabili come farebbe RUN.

Esistono però degli errori che, pur non essendo scoperti dall'interprete, non permettono al programma di eseguire ciò che noi desideriamo. Il caso classico può essere un test di IF che verifica sempre una condizione e non lascia mai eseguire una parte di istruzioni.

In questi casi, è utile il comando TRON (trace on), che permette, oltre ai normali output del programma, di visualizzare tutti i numeri di linea che il computer, di volta in volta, si trova a dover eseguire. Potremo così controllare quali linee verranno eseguite e quante volte. In questo modo osserveremo passo per passo anche tutti i salti chiesti, vedendo così quali non vengono effettuati.

Per terminare il trace basterà digitare TROFF (trace off), per tornare nella modalità normale.

# La programmazione strutturata

Non spaventatevi: non è nulla di trascendentale, è solamente un modo diverso di affrontare certe situazioni; quando vi sarete abituati ad usarla, farete fatica a staccarvene.

Cominciamo col precisare un concetto fondamentale: così come è possibile simulare (anche se con qualche problema) il compito delle subroutine e degli altri comandi di salto, lo stesso discorso vale per la programmazione strutturata. La differenza sostanziale sta nella semplicità di scrittura e lettura del programma; con essa infatti vengono tolti per quanto è possibile tutti i salti, in modo da capire, ad una prima lettura, ciò che il programma eseguirà. Inoltre risulterà più semplice sia la stesura che la correzione dello stesso.

Veniamo al primo esempio, che parte da un problema che sicuramente ognuno si è trovato a dover risolvere con qualche trucchetto. Pensiamo di dover compiere un IF e di avere due gruppi di istruzioni ben distinte da far eseguire a seconda del risultato del test.

Solitamente operavamo così:

```
10 INPUT"NUMERO";N
20 IF N<=0 THEN 60
30 PRINT"IL NUMERO E' POSITIVO"
40 LET A=A+1:LET T=T+N
50 GOTO 10
60 PRINT"TOTALE =";T
70 PRINT"HAI BATTUTO ";A;"NUMERI"
80 END
```

Ora possiamo fare:

```
10 DO
20 INPUT"NUMERO";N
```

```
30 IF N<=0 THEN EXIT
40 PRINT"IL NUMERO E' POSITIVO"
50 LET A=A+1:LET T=T+1
60 LOOP
70 PRINT"TOTALE =";T
80 PRINT"HAI BATTUTO ";A;"NUMERI"
90 END
```

In questo programmino abbiamo tre istruzioni nuove:

EXIT
DO
LOOP

L'istruzione DO determina l'inizio di un ciclo compreso tra DO e LOOP che verrà eseguito inderminatamente fino a che non incontrerà il comando EXIT, che causerà l'uscita dal ciclo per eseguire le istruzioni seguenti il LOOP.

Il programma risulta più leggibile, non esistono più i salti delle linee 20 e 50, perchè sono impliciti; non dobbiamo quindi restare legati al numero di linea alla quale saltare, ma basta posizionare al posto giusto i comandi.

Possiamo determinare l'uscita dal ciclo non solo con IF ed EXIT, ma anche con altri due comandi:

UNTIL
WHILE

Essi fungono da test come IF e determinano l'uscita come EXIT; possono però solo essere posti insieme a DO, quindi all'inizio del ciclo, o con l'istruzione LOOP, alla fine. Funzionano così: l'istruzione WHILE consente l'uscita dal ciclo solo quando la condizione del test è diventata falsa, per esempio:

```
10 DO WHILE A>0
..
90 LOOP
```

continuerà a "girare" nel ciclo fino a che A sarà minore o uguale a zero.

Con UNTIL otteniamo esattamente il contrario: continuerà ad essere eseguito il ciclo fino a che non diventerà vera la condizione.

```
10 DO UNTIL A<=0
..
90 LOOP
```

farà quindi la stessa cosa, è solamente opposta la condizione.

# Un completamento ai test: ELSE

Come abbiamo visto in uno dei primi capitoli, l'istruzione IF deve sempre essere accompagnata da THEN, che determina cosa bisogna eseguire nel caso in cui la condizione sia verificata; se la condizione non risulta valida viene eseguita la prima istruzione della linea seguente.

Possiamo però rendere ancora più semplice il test facendo stare tutto sulla stessa linea, grazie a una nuova istruzione: ELSE (altrimenti). Invece di scrivere:

```
..
40 IF A <> 0 THEN 240
50 GOTO 300
```

ora possiamo scrivere:

```
..
40 IF A <> 0 THEN 240 : ELSE 300
```

Così facendo, riusciamo a mantenere sulla stessa linea le scelte che devono essere fatte in seguito al test. Il vantaggio non sta certamente nel risparmio di una linea di programma, ma nella semplicità con cui possiamo scrivere e leggere le nostre istruzioni.

# Per finire

Da ultimo prendiamo in considerazione l'istruzione TRAP. A sentirne il nome molti programmatori potrebbero sgranare gli occhi, come se si trattasse di qualcosa di nuovo. In realtà, in altri BASIC esiste sotto il nome di ON ERROR, forse più esauriente.

Questa funzione decide il numero di linea cui dovrà saltare l'esecuzione quando incontrerà un errore. Può, ad esempio, essere utile quando vengono compiuti molti calcoli per prevedere quale errore potrà capitare e cosa sarà necessario eseguire. E' un'arma molto potente, in quanto possiamo anche sapere quale errore è avvenuto per decidere cosa fare.

Nella variabile di sistema ER è contenuto il numero corrispondente all'errore avvenuto. Compiendo dei test sul contenuto di questa variabile, possiamo far eseguire parti di programma differenti; è un altro modo per avere sempre maggiormente sotto controllo l'esecuzione del programma.

# Oltre il BASIC

Questo libro dovrebbe parlare solo di BASIC, ma ci sembra giusto fare almeno un accenno a quei comandi che stanno, per così dire, sul confine tra il BASIC e il linguaggio macchina. E' infatti possibile, attraverso dei comandi BASIC, e non solo con questi, generare dei programmi in linguaggio macchina, per esempio scrivere e leggere i contenuti nelle singole locazioni di memoria. Questa possibilità può anche essere usata per altri scopi.

I due comandi che permettono la scrittura e la lettura sono POKE e PEEK. La sintassi è praticamente identica alle normali funzioni:
POKE x,y
dove x è l'indirizzo della locazione di memoria (un numero compreso tra 0 e 65535) e y è il valore che vogliamo scrivere in quella locazione (un numero compreso tra 0 e 255).

Per PEEK, che serve per leggere il contenuto, dobbiamo solo dare l'indirizzo scritto fra parentesi.

Inoltre il BASIC 3.5 ci dà la possibilità di compiere le conversioni tra valori espressi in forma decimale e valori espressi in forma esadecimale e viceversa. Le funzioni dedicate a questo compito sono HEX$(x) e DEC(xxxx). Vediamo due esempi:

```
10 LET X$= HEX$(65535)
20 LET X= DEC (A000)
30 PRINT X$
40 PRINT X
```

In X$ troveremo la rappresentazione esadecimale di 65535 che è FFFF, in X il corrispondente decimale dell'esadecimale A000 che è 40960.

Da ultimo segnaliamo il comando SYS, che permette di mandare in esecuzione un programma scritto in linguaggio macchina. Raccomandiamo di usare con cautela questi comandi perchè, sia scrivendo che facendo eseguire programmi in linguaggio macchina, si rischia di bloccare la macchina. Nulla di grave si intende, ma se avete scritto un bel programma e provate a caso a scrivere SYS seguito da qualche numero, potreste bloccare tutto, al punto da essere costretti a resettare o addirittura a spegnere il computer perdendo il programma.

## Appendice

# Comandi e istruzioni non trattati

Ogni comando possiede una sua sintassi, in cui sono raccolte tutte le possibilità di utilizzo, oltre all'ordine e al modo con cui si devono disporre parametri ed espressioni.
Presentiamo una semplice tabella con la quale potrete leggere le carte sintattiche successive.

La parola che identifica l'istruzione o il comando sarà sempre scritta in maiuscolo (es. COPY), in minuscolo saranno invece i parametri e le espressioni.

Fra parentesi troverete tutti i parametri o le espressioni che possono essere omesse; la scritta OR indica che potete scegliere un solo parametro o elemento fra quelli scritti.

## Comandi

### BECKUP drive TO drive (,ON U numero periferica)

Questo comando permette di effettuare la copia esatta di un disco da un drive ad un altro (sono necessarie, ovviamente, due unità disco). Non è necessario che il disco di destinazione della copia sia stato formattato, viene infatti copiato anche l'header.

Esempio:

BECKUP D1 TO D0

copia il contenuto del disco presente nell'unità logica D0 nel disco inserito in D1.

**COLLECT(drive) (,ON U unità)**

Questo comando serve per cancellare i file mal chiusi (quelli con l'*), togliendo il loro riferimento dalla directory e consentendo di registrare un nuovo file al loro posto.

Esempio:

COLLECT D1

rimuove i file non chiusi dalla directory del disco inserito nel drive D1.

**COPY(drive,) "nome file" TO(drive) "altro file" (ON U unità)**

Questo comando permette di copiare un file in un altro, eventualmente cambiandone il nome, anche sullo stesso disco.

Esempio:

COPY D0,"pippo"TO D1,"pluto"

copia il file del disco presente in D0, di nome "pippo", in un altro file del disco in D1, assegnandogli il nuovo nome "pluto".

**DELETE(numero linea) (-numero linea)**

Questo comando permette di cancellare un certo numero di linee di un programma BASIC. I due parametri indicano, rispettivamente, il numero della prima e dell'ultima linea da cancellare. Ovviamente il primo numero deve sempre essere maggiore del secondo. Questo comando può solo essere usato in modo diretto, quindi non inserito in un programma.

Esempio:

DELETE 10-100

cancellerà le linee comprese tra la 10 e la 100. Nel caso, possibile, in cui venga indicato un solo parametro, verrà cancellata la sola linea menzionata.

**DIRECTORY(drive) (,U unità) (,"nome file")**

Attraverso questo comando possiamo ottenere la lista dei file presenti su un disco. E' possibile anche cercare dei file di cui conosciamo il nome, in un certo disco.

Esempio:

DIRECTORY D1

mostra la lista dei file presenti sul disco che si trova nell'unità D1.
Oppure:

DIRECTORY D1,"lol*"

visualizza tutti i file, presenti nel disco inserito in D1, il cui nome inizia con "lol".

**ONLOAD"nome file" (,drive) (,U unità)**

Permette di caricare nella memoria del computer un programma residente sull'unità a dischi.
Esempio:
DLOAD"program",D0

carica il programma "program" presente sul disco inserito in D0.

**DSAVE"nome file" (,drive) (,U unità)**

Salva su disco il programma presente nella memoria del computer.
Esempio:

DSAVE"prBASIC",D1

registra sul disco in D1 il programma giacente in memoria, assegnandogli il nome "prBASIC".

**HEADER"nome del disco", drive (,1 numero di identificazione) (,ON U unità)**

E' un comando da usare con molta attenzione! Permette di inizializzare un disco alla lettura/scrittura da parte del drive. L'operazione è necessaria su ogni disco nuovo; su un disco già scritto otteniamo la totale cancellazione di tutti i file e del nome precedente.

Esempio:

HEADER"primo disco".D0.I00

inizializza il disco inserito in D0 assegnandogli il nome "primo disco", con identificatore "00".

**KEY(tasto,stringa)**

Con questo comando possiamo assegnare ad ogni tasto di funzione (f1..f7) un comando che verrà eseguito tutte le volte che verrà in seguito premuto quel tasto.
Esempio:

KEY 1.RENUMBER+CHR$(13)

premendo il tasto f1, otterremo la rinumerazione del programma.
KEY da solo visualizza la lista dei comandi assegnati ai tasti. CHR$(13) sta al posto del tasto RETURN.

**RENAME"vecchio nome" TO "nuovo nome" (,drive) ),U unità)**

Serve per modificare il nome di un file su disco.
Esempio:

RENAME"old"to"new",D1

assegna al file "old", del disco inserito in D1, il nuovo nome "new".

**SCRATCH"nome file" (,drive) (,U unità)**

Anche in questo caso occorre molta cautela: possiamo infatti cancellare un file sul disco. Per evitare errori, il computer chiederà se siamo proprio certi di ciò che gli abbiamo chiesto di fare scrivendo ARE YOU SURE?; se voi batterete "Y" eseguirà il comando, con "N" annullerete invece la richiesta.
Esempio:

SCRATCH"pluto",D0

cancella il file di nome "pluto" dal disco presente nel drive D0.

## Le istruzioni

### BOX(numero colore,)a1,b1,(a2,b2)(,angolo)(,riempimento)

Con questo comando possiamo disegnare un rettangolo di qualunque misura, anche inclinato. Vediamo come si usano i parametri:

- a1,b1 = coordinate del punto di partenza, vertice del rettangolo;
- a2,b2 = vertice opposto al precedente, chiude il rettangolo;
- angolo = angolo di inclinazione del rettangolo;
- riempimento = determina se bordare o riempire il rettangolo: con zero disegna il contorno del colore scelto, con uno riempie il rettangolo del colore richiesto.

Esempio:

```
BOX 1,10,10,20,20
```

disegnerà il contorno di un rettangolo con il colore uno, i vertici opposti del rettangolo sono: 10,10 e 20,20.

### CHAR(numero colore),x,y,"stringa"(,reverse)

Molto simile a PRINT, questa istruzione ci permette di far stampare sullo schermo una stringa nella posizione che desideriamo, con la possibilità di scegliere il colore e il modo (reverse o normale). I valori di x,y determinano, rispettivamente, la colonna e la riga di inizio della scrittura.

Esempio:

```
CHAR 1,18,12,"stampa"
```

stampa in mezzo allo schermo la stringa "stampa" con il colore uno.

### CIRCLE(num. col.),(a,b),xr(,(yr)(,(sa)(,(ea)(,(angolo)(,inc.)))))

Dalla sintassi un po' complessa, questa istruzione è sicuramente una tra le più potenti offerte dal BASIC 3.5. Al di là dalla nostra spiegazione, vi consigliamo di sperimentare personalmente la grande versatilità di questo comando.

a,b = coordinate del centro;
xr = lunghezza del raggio;
yr = secondo raggio; permette di disegnare ellissi e figure non circolari;
sa = angolo di partenza;
ea = angolo di arrivo; se questi ultimi due parametri non completano i 360 gradi, viene disegnato un arco;
angolo = angolo di rotazione; utile per le figure non perfettamente circolari o per archi di circonferenza.
inc. = numero di gradi tra i segmenti.

Esempio:

CIRCLE,160,100,65,50,
disegnerà un cerchio

**CMD numero file**

Presente non solo nel BASIC 3.5, permette di inviare a una periferica, precedentemente aperta con il comando OPEN, l'output che normalmente inviato sullo schermo.

Esempio:

OPEN 1,4 apre il file 1 sulla stampante;
CMD 1 invia l'output dello schermo alla stampante;
LIST stampa non più su schermo ma su carta;
PRINT 1 ripristina l'output sullo schermo;
CLOSE 1 richiude il file 1.

**COLOR destinazione,numero colore(,intensità di luce)**

Possiamo variare a nostro piacimento i colori di cinque diverse sezioni dello schermo:

| numero | destinazione |
|---|---|
| 0 | sfondo |
| 1 | primo piano |
| 2 | multicolore 1 |
| 3 | multicolore 2 |
| 4 | bordo |

Esempio:

COLOR 0,1

assegna allo sfondo il colore nero.

**DRAW numero colore,a1,b1, TO a2,b2...**

Si possono tracciare delle linee congiungendo punti dello schermo con colori desiderati. Possiamo anche formare linee spezzate o figure concatenando, nella stessa istruzione, varie coppie adiacenti.
Esempio:

DRAW,10,10 TO 10,60 TO 10,10

disegnerà un triangolo.

**GETKEY lista variabili**

Come GET, assegna a una variabile stringa un carattere battuto sulla tastiera senza il tasto RETURN. A differenza di GET, il suo contenuto non può mai essere nullo, perchè il computer rimane in attesa fino a che non viene battuto un carattere. Simula la linea:

20 GETA$: IF A$=“ ” THEN 20

Esempio:

GETKY A$

Il risultato ottenuto sarà identico.

**GRAPHIC modo(,opzione di cancellamento)**

Esistono cinque modi per scrivere e disegnare sullo schermo:

| modo | descrizione |
| --- | --- |
| 0 | testo normale |
| 1 | alta risoluzione grafica |
| 2 | alta risoluzione grafica mista a testo |
| 3 | grafica multicolore |
| 4 | grafica multicolore mista a testo |

Per poterci permettere una grafica ad alta risoluzione, abbiamo bisogno di gestire lo schermo con molta più memoria di quella normalmente assegnata a tale scopo. Dovremo quindi ricordarci che ben 10 kbite di memoria non potranno essere più usati per contenere i programmi scritti in BASIC.

Esempio:

GRAPHIC 1,1

seleziona il modo grafico ad alta risoluzione e cancella lo schermo. Possiamo però riportare ad area BASIC con il comando:

GRAPHIC CLR

**LOCATE coordinata x,coordinata y**

Oltre al normale cursore lampeggiante, esiste un secondo cursore, completamente trasparente, che possiamo posizionare sullo schermo prima di disegnare.

Esempio:

LOCATE 100,100

definisce il punto dal quale verrà effettuato il primo disegno successivo.

## MONITOR

Chiamarlo comando è un po' restrittivo, in quanto sarebbe un vero e proprio programma che permette di leggere e scrivere programmi in linguaggio macchina. Il suo utilizzo sarà soprattutto apprezzato dagli utenti che già conoscono la programmazione in assembler. Dato il breve spazio a disposizione, non possiamo illustrare nessun esempio: sarebbe necessario un libro intero.

## PAINT(numero colore)(,(a,b)(,modo))

Oltre a disegnare figure, possiamo anche colorarle in maniera diversa, o colorare una certa parte di schermo.

Esempio:

```
10 CLRCLE ,100,100,65,10
20 PAINT 100,100
```

riempirà l'ellisse di colore.

## PRINT (numero file) USING lista del formato; lista di stampa

Potentissima istruzione che permette di rendere molto più elegante e professionale il comando di output PRINT. Possiamo infatti definire il formato dei numeri o dei caratteri da stampare, in modo da ottenere un perfetto ed elegante incolonnamento rispettando anche, nel caso dei numeri, la posizione del punto decimale.

Esempio:

```
50 PRINT USING"                ";48.7
```

stamperà 48.70.

Un classico esempio è una lista di numeri crescenti nella quale, a un certo punto, aumenta il numero delle cifre:

```
30 FOR I=0 TO 150
40 PRINT I;"  ";A$(I)
50 NEXT I
```

Il nostro output non sarà perfetto perchè, quando I assumerà il valore 100, la cifra decimale non sarà più incolonnata con le precedenti e tutto l'output verrà spostato di una posizione. Se invece noi digitiamo:

```
30 FOR I=0 TO 150
40 PRINT USING"             ";I;
50 PRINT " ";A$(I)
60 NEXT I
```

otteniamo l'incolonnamento di tutte le cifre.

**PUDEF"da uno a quattro caratteri"**

Possiamo modificare a piacere il formato o l'output designati dalla PRINT USING.

Esempio:

PUDEF".,"

sostituirà a ogni virgola un punto e viceversa.

**SCALE 1 or 2**

Quando stiamo operando in alta risoluzione possiamo mutare la scala di riferimento per determinare i punti dello schermo.

Esempio:

SCALE 1

aumenta il numero di punti definibili sia in alta risoluzione che in modo multicolore; si passa, infatti, da 0-319 per l'asse x e 0-199 per y in alta risoluzione, a 0-1023 per tutti e due gli assi. Con SCALE 0 ritorniamo in condizioni normali.

**SCNCLR**

Cancella tutto lo schermo, anche quello grafico.

**SOUND tono voce, parametro di frequenza, durata**

Abbiamo a disposizione 3 livelli di voce:

| Valore | Voce |
|---|---|
| 1 | voce 0(tono) |
| 2 | voce 1(tono superiore) |
| 3 | voce 3(rumore bianco, fruscio) |

Il parametro della frequenza varia da 0 a 1023 (non corrispondente agli Hz), mentre la durata va da 0 a 65535 sessantesimi di secondo.

Esempio:

SOUND 1,200,360

**VOL livello volume**

Assegna il livello di volume con cui verrà generato il comando SOUND. I valori vanno da 0 (silenzio), a 8 (volume massimo).

Esempio:

VOL 8

farà generare il prossimo comando SOUND con il volume massimo.

**WAIT indirizzo,valore1(,valore2)**

Permette di mantenere il computer in stato di attesa fino a che non sono verificate particolari condizioni nella locazione indirizzata, nélla quale vengono effettuate operazioni binarie in base ai due parametri forniti.

# Giochi e passatempi

# CONFUCIO

```
  4 CLR :PK=12
 90 PG$="HANOI":LP=1:CR$=CHR$(13)
100 T$="[RVOFF] [RVOFF]_[RVOFF]_[R
    VOFF]▁[RVOFF]▂[RVS]▔[RVS]▔[RVS
    ]▔"
105 B$="[RVS]_[RVS]_[RVS]▁[RVS]▂[R
    VOFF]▔[RVOFF]▔[RVOFF]▔[RVOFF]
    "
106 CC$="[NERO][ROSSO][AZZUR][VIOL
    A][VERDE][GIALLO][ARANC][MARR]
    [ROSA][GRIGIO1][GRIGIO2][VERDE
    2][CELESTE][GRIGIO3]"
110 DIM TP$(8),T$(8),BT$(8),B$(8),
    CO$(13)
111 FOR I=0 TO 13:CO$(I)=MID$(CC$,
```

```
    I+1,1):NEXT
120 FOR I=1 TO 8
130 C$=MID$(T$,2*I,1):C$=C$+C$+C$:
    C$=C$+C$+C$:C$=LEFT$(C$+C$,14)
140 TP$(I)=MID$(T$,2*I-1,1)+C$
150 C$=MID$(B$,2*I,1):C$=C$+C$+C$:
    C$=C$+C$+C$:C$=LEFT$(C$+C$,14)
160 BT$(I)=MID$(B$,2*I-1,1)+C$
170 NEXTI
200 L$="[RVS] [RVS]| [RVS]| [RVOFF] |
    [RVOFF] "
210 R$="[RVOFF] [RVOFF]| [RVOFF]| [R
    VS] |[RVS] "
220 DIM LF$(5),RT$(5)
230 FOR I=1 TO 5:LF$(I)=MID$(L$,2*
    I-1,2):RT$(I)=MID$(R$,2*I-1,2)
    :NEXTI
300 CR$=CHR$(13)
490 PRINT"[CLEAR]"

500 PRINT"[DOWN]QUANTI DISCONI (MA
    SSIMO 7) ? ";:GOSUB 60000
```

```
510 IF IN$="" THEN 500
560 N=VAL(IN$):IF N>7 THEN PRINT"[
    NERO]PREGO: [VIOLA]NON PIU' DI
     7.[NERO]":GOTO 500
570 IF N<2 THEN PRINT"[VERDE]NON E
    SSERE REDICOLO ![NERO]":GOTO 5
    00
900 DN$="[HOME][20 DOWN]"
910 RT$="[14 RIGHT]"
1000 FOR I=1 TO 3:FOR J=0 TO 7:P(I,
    J)=0:NEXTJ:NEXTI
1100 PRINT"[CLEAR]";DN$;"[RVS]";:P(
    1,0)=N
1110 FOR I=1 TO 3:PRINT"[ARANC]

           ";:NEXTI
1120 PRINTDN$;"[DOWN][RVS][6 RIGHT]
    #1[11 RIGHT]#2[11 RIGHT]#3[NER
    O]"
1150 PRINTDN$;LEFT$("[7 UP]",N);
1160 FOR I=1 TO N
1170 PRINTLEFT$("[RVOFF]"+RT$,7-I+1
```

```
     );LEFT$("[RVS]L"+BT$(1),2*I+1)
     ;"_|"
1180 P(1,N-I+1)=I*2
1190 NEXTI
1800 MV=0
1900 TM=TI+60
1910 IF TI<TM THEN 1910
1950 GOSUB 8000
1960 CO=CO+1:IF CO>13 THEN CO=1
1970 CO$=CO$(CO)
2000 PR$="[NERO]DA QUALE TORRE ? ":
     GOSUB 5000:IF IN$="" THEN 1950
2010 F=VAL(IN$)
2020 IF P(F,0)<1 THEN PRINT"[VERDE2
     ]TORRE VUOTA [ROSSO]!![NERO]":
     GOTO 1900
2050 PR$="[NERO]ALLA TORRE ? ":GOSU
     B 5000:IF IN$="" THEN 1950
2060 T=VAL(IN$)
2070 IF F=T THEN PRINT"MOSSA IMPOSS
     IBILE ":GOTO 1900
```

```
2080 IF P(T,0)=0 THEN 2100
2090 IF P(F,P(F,0))>P(T,P(T,0)) THE
     N PRINT"[VERDE2]NON PUOI[NERO]
     ":GOTO 1900
2100 FC=P(F,0):FW=P(F,FC):TC=P(T,0)
     :TW=P(T,TC):MV=MV+1
2110 X=1+13*(F-1)+7-FW/2
2120 PRINT DN$;LEFT$("[9 UP]",FC+1)
     ;LEFT$("[RVOFF]"+RT$+RT$+RT$,X
     );
2125 LF$=LEFT$("[DOWN][14 LEFT]",FW
     +1)
2130 FOR I=1 TO 8:T$(I)=LEFT$(TP$(I
     ),1+FW):B$(I)=LEFT$(BT$(I),1+F
     W):NEXT
2135 HT=0:J=1:IF F>T THEN J=-1
2140 FOR I=F TO T STEP J
2145 IF P(I,0)>HT THEN HT=P(I,0)
2150 NEXTI
2155 IF HT=P(T,0) THEN HT=HT+1:GOTO
      2165
2160 IF ABS(F-T)>1 THEN IF HT=P(2,0
```

```
     ) THEN HT=HT+1
2165 FOR I=FC TO HT
2170 :FOR  J=1 TO 8
2175 ::PRINT CO$;        T$(J);LF$;B$(
     J);LF$;"[2 UP]";
2180 :NEXTJ
2185 :PRINT"[UP]";
2190 NEXTI
2195 PRINT"[DOWN]";
2200 L$=LEFT$("[16 LEFT]",1+FW):R$=
     LEFT$(RT$,FW-1)
2201 PRINT"[RVS]";LEFT$("
           ",FW);"[RVOFF]";LEFT$("[
     16 LEFT]",FW);
2205 IF F>T THEN 2250
2210 FOR I=F*13 TO T*13-1
2215 :FOR J=1 TO 5
2220 PRINT CO$; LF$(J);R$;RT$(J);L$
     ;
2225 NEXTJ
2226 PRINT"[RIGHT]";
```

```
2230 NEXTI
2235 GOTO 2300
2250 FOR I=T*13 TO F*13-1
2255 PRINT"[LEFT]";
2260 :FOR J=5 TO 1 STEP -1
2265 ::PRINT CO$;     LF$(J);R$;RT$(
     J);L$;
2270 :NEXTJ
2275 NEXTI
2300 FOR I=HT-1 TO TC STEP -1
2310 :FOR J=8 TO 1 STEP -1
2320 :PRINT CO$;  T$(J);LF$;B$(J);L
     F$;"[2 UP]";
2330 :NEXTJ
2340 :PRINT"[DOWN]";
2350 NEXTI
2410 PRINT CO$;  LEFT$("[RVS]L"+BT$
     (1),FW+1);"_|";  "[NERO]";
2500 P(T,0)=P(T,0)+1
2510 P(T,P(T,0))=P(F,P(F,0))
2520 P(F,0)=P(F,0)-1
```

```
2600 IF P(2,0)<>N AND P(3,0)<>N THEN 1900
2700 GOSUB 8000
2710 PRINT"[VERDE][3 DOWN]                    ▪ ████████████████████"
2720 PRINT"          [RVS]▌HAI FINITO [RVOFF]▌"
2730 PRINT"          ▪[RVS]██████████████[RVOFF]▪"
2740 PRINT"[NERO][DOWN]HA IMPIEGATO [VIOLA]";MV;"[NERO]MOSSE"
2750 T=2↑N-1:PRINT"[DOWN]LA SOLUZIONE PIU' VELOCE"
2760 PRINT"CORRISPONDE A ";T;"MOSSE."
2770 PRINT"[DOWN]GIOCHI ANCORA? ";:GOSUB 60000
2780 IF LEFT$(IN$,1)<>"S" THEN END
2790  RUN
2800 END
5000 PRINTPR$;
```

```
5010 GOSUB 60000:IF IN$="" THEN RET
     URN
5015 IF LEFT$(IN$,1)="Q" THEN END
5016 IF LEN(IN$)>1 THEN 5030
5020 IF IN$>="1" AND IN$<="3" THEN
     RETURN
5030 PRINT"MA COSA DICI ????????"
5040 FOR I=1 TO 500:NEXTI
5050 PRINT"[UP]
      "
5060 PRINT"[2 UP]";PR$;LEFT$("


          ",LEN(IN$))
5070 PRINT"[UP]";:GOTO 5000
8000 PRINT"[HOME]";
8010 FOR I=1 TO 4:PRINT"[BIANCO]


           ":NEXTI
8020 PRINT"[HOME]";
8030 RETURN
9000 PRINT"[HOME]"
```

```
 000 IN$=" ":ZT=TI:ZC=2:ZD$=CHR$(20
     )
 010 GET Z$:IF  Z$<>"" THEN 60070
 020 IF ZT<=TI THEN PRINTMID$(" ▒",
     ZC,1);"[LEFT]";:ZC=3-ZC:ZT=TI+
     15
 030 GOTO 60010
60070 Z=ASC(Z$):ZL=LEN(IN$):IF (Z AN
     D 127)<32 THEN PRINT" [LEFT]";
     :GOTO 60110
60080 IF FL AND (Z AND 127)>64 AND (
     Z AND 127)<91 THEN Z$=CHR$((Z+
     128) AND 255)
60090 IF ZL>254 THEN 60010
60100 IN$=IN$+Z$:PRINTZ$;ZD$;Z$;
60110 IF Z=13 THEN IN$=MID$(IN$,2):P
     RINTCR$;:RETURN
60120 IF Z=20 AND ZL>1 THEN IN$=LEFT
     $(IN$,ZL-1):PRINT"[LEFT]";:GOT
     O 60010
60130 IF Z=141 THEN Z$=CHR$(-20*(ZL>
```

```
    1)):FOR Z=2 TO ZL:PRINTZ$;:NEX
    TZ:GOTO 60000
60140 GOTO 60010
```

# 9 CARTE

```
100 REM  GIOCO DELLE 9 CARTE
110 :
120 REM  BY FLAVIO MOLINARI
125 :
130 REM  C64 - C16
135 :
140 REM  ***************************

142 REM  VARIABILI IN USO
144 :
146 REM  Q,K,H,J,Z=VARIABILI DI CO
    MODO
148 REM  D$()=POSIZIONE CURSORE
150 REM  C$()=CARTE
152 REM  C8$ =CANCELLA CARTA
154 REM  PG  =GIOCATORE CHE INIZIA
155 REM  LE  =LIVELLO
156 REM  T1()=CARTE DEL GIOCATORE
158 REM  T2()=CARTE DEL COMPUTER
```

```
160 REM  T3()=CARTE IN TAVOLA
162 REM  N1  =NUM. PESCATE GIOCATO
    RE
164 REM  N2  =NUM. PESCATE COMPUTE
    R
166 REM  PA$ =MEMORIZZA PARTITA
168 REM  GC  =GIOCATORE DI TURNO:
    1=GIOC.  2=COMPUTER
170 REM  G   =CARTA PESCATA
172 REM  A   =SCELTA TEMPORANEA CO
    MPUTER
174 REM  U   =VARIABILE SCELTA PSE
    UDOCASUALE
180 REM  ***************************
210 DIM D$(24):D$(1)=CHR$(19)
220 FOR Q=2 TO 24:D$(Q)=CHR$(19
    )+D$(Q-1)+CHR$(17):NEXT
230 C2$="               ":REM 15
    SPAZI
498 REM  ***************************
500 REM       PREPARA CARTE
502 REM  ***************************
510 C0$=CHR$(38):C1$=CHR$(17)+CHR$
    (157)+CHR$(157)+CHR$(157)
520 C3$=CHR$(117)+CHR$(96)+CHR$(10
    5)+C1$:C4$=C0$+CHR$(32)+C0$+C1
    $
```

```
540 C6$=CHR$(106)+CHR$(96)+CHR$(10
    7):C7$="   ":REM  3 SPAZI
560 FOR  Q=1  TO  5:C8$=C8$+C7$+C1
    $:NEXT
570 FOR  Q=1  TO  9:Q$=MID$(STR$(Q
    ),2,1)
580 C$(Q)=C3$+C4$+C0$+Q$+C0$+C1$+C
    4$+C6$:NEXT
698 :
700 PG=RND(1)*2-1:REM  SCEGLIE CHI
     INIZIA
798 REM  **************************
800 REM          INIZIALIZZA
802 REM  **************************
810 FOR  Q=1  TO  9:T1(Q)=0:T2(Q)=
    0:T3(Q)=Q:NEXT
820 N1=0:N2=0:REM  NUMERO CARTE GI
    OCATE
830 PA$="":REM  MEMORIZZA PARTITA
850 PRINT CHR$(147)SPC(252)"LIVELL
    O (1/2)";:INPUT  LE
860 IF  LE<1  OR  LE>2  THEN  850
998 REM   *************************
1000 REM        DISPONE CARTE
2010 FOR  K=1  TO  N2-1:FOR  H=K+1
      TO  N2
```

```
2020 A=15-T2(K)-T2(H):IF  A>9  OR
     A<0  THEN  A=0
2030 IF  T3(A)=0  THEN  2050:REM  G
     IA' GIOCATA
2040 G=A:V1=G:V2=T2(K):V3=T2(H):REM
      COMB. VINCENTE
2050 NEXT:NEXT
2060 IF  G<>0  THEN  GOSUB  6010:GO
     SUB  7510:GOTO  5010:REM  SI
2198 REM  ************************
2200 REM       GIOCATA FORZATA ?
2202 REM  ************************
2205 IF  N1<2  THEN  2250:REM  NO
2210 FOR  K=1  TO  N1-1:FOR  H=K+1
      TO  N1
2220 A=15-T1(K)-T1(H):IF  A>9  OR
     A<0  THEN  A=0
2230 IF  T3(A)<>0  THEN  G=A
2240 NEXT:NEXT
2250 IF  G<>0  THEN  GOSUB  6010:GO
     TO  1410
2398 REM  ************************
2400 REM   STRATEGIA DEL COMPUTER
2402 REM  ************************
2410 G=0:IF  LE=2  THEN  GOSUB  100
     10:REM  STRATEGIE LIV 2
```

```
2420 IF  G<>0  THEN  2490:REM  HA G
     IA' SCELTO
2430 U=2:IF  N1+N2>5  THEN  U=1
2450 G=INT(RND(1)*10/U)*U:REM  SCEL
     TA PSEUDOCASUALE
2460 IF  G=0  THEN  G=5
2470 IF  T3(G)=0  THEN  2450:REM  G
     IA' GIOCATA
2490 GOSUB  6010:GOTO  1410:REM  TO
     RNA AL GIOCATORE
4998 REM  ***********************
5000 REM        FINE DELLA PARTITA
5002 REM  ***********************
5005 PRINT D$(22) TAB(20)"PARTITA P
     ARI."
5010 GOSUB  8000:PRINT D$(23) TAB(2
     0)"GIOCHIAMO ANCORA ?"
5020 GET  Q$:IF  Q$="S"  THEN  PG=-
     PG:GOTO  810
5030 IF  Q$="N"  THEN  END
5040 GOTO  5020
5998 REM  ***********************
6000 REM         SPOSTA CARTA E
6002 REM        AGGIORNA PARTITA
6004 REM  ***********************
6010 PRINT D$(11) TAB(G*3+3)C8$:T3(
```

```
     G)=0:REM  CANCELLA CARTA
6020 IF  GC=1  THEN  PRINT D$(19) T
     AB(N1*3)C$(G):N1=N1+1:T1(N1)=G
6030 IF  GC=2  THEN  PRINT D$( 2) T
     AB(N2*3)C$(G):N2=N2+1:T2(N2)=G
6040 PA$=PA$+MID$(STR$(G),2,1):RETU
     RN
6998 REM  ************************
7000 REM       VINCE IL GIOCATORE
7002 REM  ************************
7010 FOR  Z=1  TO  N1
7020 IF  T1(Z)=V1  OR  T1(Z)=V2  OR
       T1(Z)=V3  THEN  7040
7030 GOSUB  8000:PRINT D$(19) TAB(Z
     *3-3)C8$:REM  CANCELLA CARTA
7040 NEXT
7060 PRINT D$(22) TAB(20)"BRAVO, HA
     I VINTO !":RETURN
7498 REM  ************************
7500 REM       VINCE IL COMPUTER
7502 REM  ************************
7510 FOR  Z=1  TO  N2
7520 IF  T2(Z)=V1  OR  T2(Z)=V2  OR
       T2(Z)=V3  THEN  7540
7530 GOSUB  8000:PRINT D$( 2) TAB(Z
     *3-3)C8$
```

```
7540 NEXT
7560 PRINT D$(22) TAB(20)"HO VINTO
     IO !!":RETURN
7398 :
8000 FOR  Q=1  TO  2000:NEXT:RETURN
     :REM  RITARDO
9398 REM    *************************
10000 REM   STRATEGIE LIVELLO 2
10002 REM  *************************
10010 PA=VAL(PA$)
10020 IF  PA=224  OR  PA=226  THEN
      G=8
10030 IF  PA=242  OR  PA=248  THEN
      G=6
10040 IF  PA=262  OR  PA=268  THEN
      G=4
10050 IF  PA=286  OR  PA=284  THEN
      G=2
10200 IF  PA=1258  OR  PA=1852  OR
      PA=1456  OR  PA=1654  THEN  G=
      7
10300 IF  G=0  AND  T3(5)<>0  AND  N
      1+N2>0  THEN  G=5
11000 RETURN
```

# Incremento demografico

```
100 REM  *************************
    ****
110 REM  *
       *
120 REM  *  INCREMENTO DEMOGRAFICO
       *
130 REM  *
       *
140 REM  * COMMODORE 64, C-16, PLU
    S4 *
150 REM  *
       *
160 REM  *     DI  LUCA GALUZZI
       *
170 REM  *
       *
180 REM  *************************
```

```
     ****
190  :
205 PRINT"[CLEAR]1- COMMODORE 64"
206 PRINT"2- C-16 & PLUS 4"
207 GET A$:IF A$="1" THEN SC=1024:
    GOTO 210
208 IF A$="2" THEN SC=3072:GOTO 21
    0
209 GOTO 207
210 PRINT  CHR$(147)
220 PRINT"[3 DOWN]PER QUANTO TEMPO
     VUOI CHE DURI LA"
230 PRINT"[DOWN]SIMULAZIONE ?"
240 PRINT"[DOWN]INSERISCI IL TEMPO
     LIMITE NELLA FORMA"
250 PRINT"[DOWN]ORE MINUTI SECONDI
    "
260 INPUT "[DOWN]OOMMSS";T$
270 IF LEN(T$)<>6 THEN 210
280 TF$=LEFT$(T$,2)+":"+MID$(T$,3,
    2)+":"+RIGHT$(T$,2)
290 PRINT  CHR$(147)
300 REM  I VARI SIMBOLI GRAFICI
310 REM  SONO OTTENUTI CON I TASTI
    :
320 REM     B,U,I,J,K,C + SHIFT
```

```
360 :
370 PRINT" ┌──────────────────────────────────────────
    ┬───────────────────"
380 PRINT"|
    |                   "
390 PRINT"| ETA'      %     %  POPOLAZ.
    |   CONTROLLO:      "
400 PRINT"|                    MONDIALE
    |                   "
410 PRINT"|
    |  M      E     F   "
420 PRINT"| 0-15
    |  O      T     I   "
430 PRINT"|
    |  R      A'    G   "
440 PRINT"|15-30
    |  T            L   "
450 PRINT"|
    |  A      D     I   "
460 PRINT"|30-50
    |  L      I         "
470 PRINT"|
    |  I            P   "
480 PRINT"|50-70
    |  T      M     E   "
490 PRINT"|
```

```
    |  A'     A      R  "
500 PRINT"|70-99
    |         T         "
510 PRINT"|
    |  I      R      C  "
520 PRINT"|            TOTALE:
    |  N      I      O  "
530 PRINT"|
    |  F      M      P  "
540 PRINT"|
    |  A      O      P  "
550 PRINT"|
    |  N      N      I  "
560 PRINT"|
    |  T      I      A  "
570 PRINT"|
    |  I      O         "
580 PRINT"|
    |  L                "
590 PRINT"|
    |  E                "
600 PRINT"└──────────────────────────────
    ┴───────────────────"
610 REM  PERCENTUALI  INIZIALI
620 E1=INT(RND(1)*10)+18
630 E2=INT(RND(1)*10)+25
```

```
640 E3=INT(RND(1)*10)+23
650 E4=INT(RND(1)*10)+16
660 E5=100-(E1+E2+E3+E4):IF E5<0 T
    HEN 620
670 A$="[5 DOWN][7 RIGHT]"
680 B$="[DOWN][7 RIGHT]"
690 REM  STAMPA PERCENTUALI INIZIA
    LI
700 PRINT"[HOME]";A$;E1
710 PRINTB$;E2
720 PRINTB$;E3
730 PRINTB$;E4
740 PRINTB$;E5
750 REM  INIZIALIZZA MORTALITA',ET
    A' DI MATR. E # FIGLI
760 MO=1:MA=1:FI=1
770 REM  CALCOLA POPOLAZIONE INIZI
    ALE
780 Q1=100*E1:Q2=100*E2:Q3=100*E3:
    Q4=100*E4:Q5=100*E5
790 REM  RESETTA L'OROLOGIO
800 TI$="000000"
810 PRINT"[HOME]";A$;A$;A$;A$
820 PRINT"[UP][3 RIGHT]LIMITE:";TF
    $
830 REM  STAMPA L'OROLOGIO FORMATT
```

```
    ATO
840 PRINT"[3 RIGHT]TEMPO:[RIGHT]";
    LEFT$(TI$,2);":";MID$(TI$,3,2)
    ;":";RIGHT$(TI$,2)
850 PRINT"[HOME]";
860 REM  INCOLONNAMENTO A DESTRA
870 PRINT TAB(17-LEN(STR$(INT(Q1))
    ))A$;"[LEFT]    [2 LEFT]";INT(Q
    1)
880 PRINT TAB(17-LEN(STR$(INT(Q2))
    ))B$;"[LEFT]    [2 LEFT]";INT(Q
    2)
890 PRINT TAB(17-LEN(STR$(INT(Q3))
    ))B$;"[LEFT]    [2 LEFT]";INT(Q
    3)
900 PRINT TAB(17-LEN(STR$(INT(Q4))
    ))B$;"[LEFT]    [2 LEFT]";INT(Q
    4)
910 PRINT TAB(17-LEN(STR$(INT(Q5))
    ))B$;"[LEFT]    [2 LEFT]";INT(Q
    5)
920 KK=QT:QT=INT(Q1)+INT(Q2)+INT(Q
    3)+INT(Q4)+INT(Q5)
930 PRINT TAB(17-LEN(STR$(INT(QT))
    ))B$;"[LEFT]    [2 LEFT]";INT(Q
    T)
```

```
 940 IF TI$=T$ THEN 1680
 950 REM  STAMPA LA COLONNINA DELLA
      MORTALITA'
 960 X=26:Y=22-MO:POKE SC+X+40*Y,32
 970 FOR Y=22 TO 23-MO  STEP -1
 980 POKE SC+X+40*Y,97
 990 NEXT
1000 REM  STAMPA LA COLONNINA DELL'
     ETA' DI MATR.
1010 X=31:Y=22-MA:POKE SC+X+40*Y,32
1020 FOR Y=22 TO 23-MA  STEP -1
1030 POKE SC+X+40*Y,97
1040 NEXT
1050 REM  STAMPA LA COLONNINA DEL N
     UM. FIGLI
1060 X=36:Y=22-FI:POKE SC+X+40*Y,32
1070 FOR Y=22 TO 23-FI  STEP -1
1080 POKE SC+X+40*Y,97
1090 NEXT
1100 REM  INPUT TASTI DI CONTROLLO
1110 REM  'Q' E 'A' PER LA MORTALIT
     A'
1120 REM  'W' E 'S' PER L'ETA' DI M
     ATR.
1130 REM  'E' E 'D' PER IN NUM. FIG
     LI
```

```
1140 GET D$
1150 IF D$="Q" THEN MO=MO-(MO<19)
1160 IF D$="A" THEN MO=MO+(MO>1)
1170 IF D$="W" THEN MA=MA-(MA<19)
1180 IF D$="S" THEN MA=MA+(MA>1)
1190 IF D$="E" THEN FI=FI-(FI<19)
1200 IF D$="D" THEN FI=FI+(FI>1)
1210 REM  INCREMENTO E DECREMENTO D
     ELLE
1220 REM  SINGOLE FASCE DI ETA'
1230 IF Q1<=0 THEN 1260
1240 Q1=(Q1-(MO/5))
1250 Q1=(Q1-(MA/30))
1260 Q1=(Q1+((FI/80)*(Q2/400)*(Q3/4
     00)))
1270 IF Q1<1 THEN Q1=0
1280 IF Q1<Q2 THEN Q2=Q2-Q2/300
1290 IF Q1<2*Q2 THEN Q2=Q2-Q2/700
1300 IF Q1<3*Q2 THEN Q2=Q2-Q2/1500
1310 Q2=Q2+Q1/100
1320 IF Q2<Q3 THEN Q3=Q3-Q3/100
1330 IF Q2<2*Q3 THEN Q3=Q3-Q3/ 500
1340 Q3=Q3+Q2/500
1350 IF Q3<Q4 THEN Q4=Q4-Q4/100
1360 IF Q3<2*Q4 THEN Q4=Q4-Q4/ 500
1370 Q4=Q4+Q3/1000
```

```
1380 IF Q4<Q5 THEN Q5=Q5-Q5/100
1390 IF Q4<2*Q5 THEN Q5=Q5-Q5/ 500
1400 Q5=Q5+Q4/2000
1410 REM  QT=TOTALE POPOLAZIONE
1420 IF QT>90000000 THEN GOSUB 1600
1430 Z=QT/100
1440 REM  CALCOLO EFFETTIVE PERCENT
     UALI POPOLAZIONE
1450 P1=INT(Q1/Z):P2=INT(Q2/Z)
1460 P3=INT(Q3/Z):P4=INT(Q4/Z)
1470 P5=INT(Q5/Z)
1480 H$="[4 RIGHT]"
1490 REM  STAMPA PERCENTUALI
1500 PRINT"[HOME]";H$;A$;"    [3 LEF
     T]";INT(P1)
1510 PRINTB$;H$;"    [3 LEFT]";INT(P
     2)
1520 PRINTB$;H$;"    [3 LEFT]";INT(P
     3)
1530 PRINTB$;H$;"    [3 LEFT]";INT(P
     4)
1540 PRINTB$;H$;"    [3 LEFT]";INT(P
     5)
1550 PRINTA$;"[4 LEFT]PUNTEGGIO :
          [6 LEFT]";INT(PP)
1560 REM  PUNTEGGIO
```

```
1570 IF ABS(QT-KK)<3 THEN PP=PP+QT/
     100:GOTO 810
1580 PP=0
1590 GOTO 810:REM  CONCLUDE IL CICL
     O
1600 REM  ************************
1610 REM  *** SOVRAPOPOLAZIONE ***
1620 REM  ************************
1630 Q1=Q1/7
1640 Q5=Q5/7
1650 Q4=Q4/5
1660 Q2=Q2/4
1670 RETURN
1680 REM  *************************
1690 REM  ********  FINE  *********
1700 REM  *************************
1710 PRINTCHR$(147)
1720 PRINT"[3 DOWN]"
1730 PRINT"HAI TOTALIZZATO ";INT(PP
     );" PUNTI"
1740 PRINT"[3 DOWN]"
1750 PRINT"CON UNA POPOLAZIONE DI "
     ;QT;" ANIME"
1760 PRINT"[3 DOWN]"
1770 PRINT"[UP]  VUOI RICOMINCIARE
     (S/N)"
```

```
1780 GET A$:FOR X=1 TO 100:NEXT
1790 PRINT"[UP]
               "
1800 FOR X=1 TO 100:NEXT:IF A$="" T
     HEN 1770
1810 IF A$="S" THEN 200
1820 END
```

# Simulazione televideo

```
100 REM  COMMODORE 64, C-16
110 REM  PLUS-4, PET 4000-8000
115 :
120 REM  SIMULAZIONE TELEVIDEO
160 REM  BY GIOVANNI BELLU'
230 :
360 INPUT "[CLEAR]DATA ODIERNA (ES
    .20/08/1985)";DT$:IF  LEN(DT$)
    <>10 THEN 360
365 INPUT "CHE ORA E' (HHMMSS)";T$
    :IF LEN(T$)<>6 THEN 365
370 TI$=T$
380 PRINT"[CLEAR][RVS] HAI :"
390 PRINT"1-C= 64"
400 PRINT"2-C= 16 O PLUS 4"
410 GET A$:IF A$<"1" OR A$>"2" THE
    N 410
```

```
420 A=VAL(A$)
430 ON  A GOSUB  690,760
440 P1$="[HOME][DOWN][3 RIGHT]"
450 P2$="[HOME]":FOR K=1 TO 31:P2$
    =P2$+"[RIGHT]":NEXT
460 P2$=P2$+"[GIALLO]"
470 P1$=P1$+"[VERDE]"
480 NP=0:REM  NUMERO PAGINA
490 GOSUB 1130:GOSUB 880
500 W$=STR$(NP):W$="000"+RIGHT$(W$
    ,LEN(W$)-1)
510 W$=RIGHT$(W$,3)
520 PRINTP1$W$:PRINTP2$TI$
530 P=PEEK(F)
540 IF P=A THEN NP=NP+1:IF NP>999
    THEN NP=0
550 IF P=B THEN NP=NP-1:IF NP<0 TH
    EN NP=999
560 IF P=C THEN NP=NP+10:IF NP>999
     THEN NP=0

570 IF P=D THEN NP=NP-10:IF NP<0 T
    HEN NP=999
580 IF P=E THEN 600

590 GOTO 500
600 W$=STR$(NP):W$="000"+RIGHT$(W$
```

```
    ,LEN(W$)-1)
610 W$=RIGHT$(W$,3)
620 PRINTP1$W$:PRINTP2$TI$
630 IF NP=0 THEN GOSUB 1130:GOSUB
    880:GOTO 500
640 IF NP<101 THEN GOSUB 1130:GOTO
     1180
650 IF NP<301 THEN GOSUB 1130:GOTO
     1330
660 IF NP<406 THEN GOSUB 1130:GOTO
     1520
670 GOSUB 1130:GOSUB 1680
680 END
690 REM ****************************
    *
700 REM *INIT.VARIABILI PER C= 64
    *
710 REM ****************************
    *
720 POKE 53280,0:POKE 53281,0:REM
     COLORE SFONDO + BORDO = NERO
730 A=5:B=6:C=4:D=3:E=1:F=197
740 PRINT"[CLEAR][BIANCO]"
750 RETURN
```

```
760 REM ****************************
    *
770 REM *INIT.VARIABILI PER C= 16
    *
772 REM ****************************
    *
780 REM  COLORE DEL FONDO, BORDO,
    CARATTERE
785 REM  DIGITARE LA LINEA 790 SEN
    ZA L'ISTRUZIONE "REM"
790 REM : COLOR 0,1: COLOR 4,1: CO
    LOR 1,1
800 A=5:B=6:C=4:D=3:E=1:F=198
810 PRINT"[CLEAR][BIANCO]"
820 RETURN
830 REM ****************************
    *
840 REM *PAGINA 000 INDICE PAGINE
    *
850 REM ****************************
    *
860 PRINT"[CLEAR][BIANCO]"
870 RETURN
872 :
875 REM  SCRIVE "RAI"
```

```
 880 PRINT"[VERDE]
                                    ";
 830 PRINT"  ▒▒▒   ▒▒▒▒  ▒
                         ";:REM  TASTO
     COMMODORE & "+"
 900 PRINT"  ▒  ▒  ▒  ▒  ▒
1160 RETURN              ";:REM  TASTO
1170 REM *****************************
1180 REM *       DATI ULTIM'ORA
1190 REM *****************************
     *
1200 PRINT"[ROSSO][RVS] [RVOFF] PER
     ICOLO IN MARE:
          ";
1210 PRINT"[GIALLO]UN PICCOLO SQUAL
     O E' STATO VISTO DA UN  ";
1220 PRINT"BAGNANTE A VARAZZE, UNA
     LOCALITA'         ";
1230 PRINT"BALNEARE LIGURE.
                         ";
1240 PRINT"SPAVENTATO SI E' SUBITO
     PRECIPITATO A   ";
1250 PRINT"RIVA, SOCCORSO PRONTAMEN
     TE DAL BAGNINO. ";
1260 PRINT"PIU' TARDI SI E' SCOPERT
     O CHE ERA SOLO  ";
1270 PRINT"UNO SCHERZO DI UN RAGAZZ
     INO, E LA GENTE ";
```

```
1280 PRINT"HA POTUTO CONTINUARE LE
     NUOTATE.                ";
1290 PRINT"MANCAVA SOLO QUESTO PER
     VIVACIZZARE UN  ";
1300 PRINT"PO' LE VACANZE.
                             ";
1310 GOTO 500
1320 REM ***************************
     *
1330 REM *       DATI SPORTIVI
     *
1340 REM ***************************
     *
1350 IF NP=157 THEN GOTO 1840
1360 PRINT"[ROSSO][RVS] [RVOFF][VIO
     LA]INCREDIBILE:[AZZUR] RUMENIG
     GE LASCIA L'INTER. ";
1370 PRINT"
                             ";
1380 PRINT"[CELESTE]DISACCORDI SUL
     PIANO FINANZIARIO HANNO  ";
1390 PRINT"FATTO DECIDERE A RUMENIG
     GE CHE NON POTE-";
1400 PRINT"VA PIU' STARE NELLA SQUA
     DRA MILANESE.   ";
1410 PRINT"FORSE VERRA' INGAGGIATO
```

```
     DAL MILAN?          ";
1420 PRINT"
                         ";
     COMMODORE & "-"
 910 PRINT"  ▒▒▒  ▒▒▒▒ ▒
                         ";
 920 PRINT"  ▒  ▒  ▒  ▒  ▒
                         ";
 930 PRINT"
                         ";
 935 :
 936 REM  SCRIVE "TELEVIDEO"
 940 PRINT"[VERDE2]  ▒▒▒▒ ▒▒▒ ▒  ▒
     ▒▒ ▒  ▒ ▒ ▒▒  ▒▒▒ ▒▒▒   ";
 950 PRINT"  ▒  ▒▒  ▒  ▒▒  ▒  ▒
     ▒ ▒ ▒ ▒▒  ▒ ▒   ";
 960 PRINT"  ▒  ▒  ▒  ▒   ▒▒
     ▒ ▒ ▒ ▒   ▒ ▒   ";
 970 PRINT"  ▒  ▒▒▒ ▒▒▒ ▒▒▒  ▒▒
     ▒ ▒▒  ▒▒▒ ▒▒▒   ";
 980 PRINT"
                         ";
 990 PRINT"  [ROSSO]INDICE DATI DI
     OGGI                ";
1000 PRINT"
                         ";
1010 PRINT"[VIOLA]  PAGINA 000 = IN
```

```
     DICE                          ";
1020 PRINT"  DA 001 A 100 ULTIM'ORA
                          ";
1030 PRINT"  DA 101 A 300 SPORT
                          ";
1040 PRINT"  DA 301 A 405 ECONOMIA
                          ";
1050 PRINT"  DA 406 A 999 VARI
                          ";
1060 PRINT"
                          ";
1070 PRINT"
                          ";
1080 PRINT"[HOME]"
1090 RETURN
1100 REM ****************************
1110 REM * VISUALIZZA INTESTAZIONE
     *
1120 REM ****************************
     *
1130 PRINT"[CLEAR]PAGINA     DATA:"D
     T$"  ORA:"TI$
1140 PRINT"    [VERDE]000     [VIOLA]
     RAI SERVIZIO TELEVIDEO[BIANCO]
          ";
1150 PRINT"[AZZUR]────────────────
     ────────────────────────";
```

```
1430 PRINT"[RVS][ROSSO] [RVOFF][VER
     DE] IN DUBBIO LA PARTECIPAZION
     E DI SARONNI";
1440 PRINT"AI PROSSIMI CAMPIONATI M
     ONDIALI DI         ";
1450 PRINT"CICLISMO SU STRADA.
                        ";
1460 PRINT"                 ";
1470 PRINT"[RVS][ROSSO] [RVOFF][VER
     DE2] IN OTTIMA FORMA INVECE E'
      IL TRENTINO ";
1480 PRINT"MOSER,CHE SARA' COME ORM
     AI DA MOLTI ANNI";
1490 PRINT"UNO DEI PIU' FAVORITI FR
     A GLI AZZURI.   ";
1500 GOTO 500
1510 REM ************************
1520 REM *     DATI ECONOMICI    *
1530 REM ************************
1540 PRINT"[RVS][ROSSO] [RVOFF][VER
     DE2][BIANCO] SALGONO LE QUOTAZ
     IONI DELLA SYSTEMS    ";
1550 PRINT"
                        ";
1560 PRINT"[RVS][ROSSO] [RVOFF][VER
     DE2] IN SETTIMANA AUMENTI DI B
```

```
     ENZINA E       ";
1570 PRINT"  GASOLIO.
                        ";
1580 PRINT"
                        ";
1590 PRINT"[RVS][ROSSO] [RVOFF][VER
     DE2] GIORNATA CALMA PER LE MON
     ETE EUROPEE. ";
1600 PRINT"
                        ";
1610 PRINT"
                        ";
1620 PRINT"[RVS][ROSSO] [RVOFF][VER
     DE2] GLI ESPERTI PREVEDONO UN
     VISTOSO        ";
1630 PRINT"  AUMENTO DELL'ORO.
                        ";
1640 PRINT"
                        ";
1650 PRINT"[RVS][ROSSO] [RVOFF][VER
     DE2] AUMENTANO I TASSI DI INTE
     RESSE USA    ";
1660 GOTO 500
1670 REM **************************
1680 REM *       DATI VARI        *
1690 REM **************************
```

```
1700 PRINT"[RVS][ROSSO] [RVOFF][VER
     DE2][AZZUR] RICETTA DEL GIORNO
     : LA PIZZA.          ";
1710 PRINT"
                          ";
1720 PRINT"[GRIGIO1]INGREDIANTI:
                                   ";
1730 PRINT"[GRIGIO3]FARINA,LIEVITO,
     POMODORI,MOZZARELLA,SALE ";
1740 PRINT"ORIGANO,OLIO,PROSCIUTTO
     E ... FORNO.      ";
1750 PRINT"
                          ";
1760 PRINT"[VERDE]PREPARAZIONE:[VER
     DE2]UNITE FARINA ED ACQUA, POI
      ";
1770 PRINT"AGGIUNGETE IL LIEVITO ED
      IL SALE.         ";
1780 PRINT"LASCIARE LIEVITARE PER A
     LMENO UN ORA.     ";
1790 PRINT"INFINE STENDETE LA PASTA
      E GUARNITELA     ";
1800 PRINT"CON MOZZARELLA POMODORO
     E PROSCIUTTO.     ";
1810 PRINT"METTETE IN FORNO PER 20
     MINUTI.           ";
1820 PRINT"ALLA FINE GUSTERETE UNA
```

```
     OTTIMA PIZZA.    ";
1830 GOTO 500
1840 REM ************************
1850 REM *       PAGINA 157      *
1860 REM *PAGINA DI..FINE SCHERZO*
1870 REM ************************
1880 PRINT"[RVS][ROSSO] [RVOFF][VER
     DE2] SEI PROPRIO UN POLLO
                  ";
1890 PRINT"
                     ";
1900 PRINT"[ROSA]CI SEI CASCATO IN
     PIENO.                ";
1910 PRINT"
                     ";
1920 PRINT"[BLEU]MA QUANDO IMPARERA
     I A NON ESSERE COSI'  ";
1930 PRINT"
                     ";
1940 PRINT"CREDULONE ???!!.....
                     ";
1950 GOTO 500
1960 REM  GIOVANNI BELLU' SOFTWARE
     1985
```

I LIBRI DI SYSTEM - libro mensile di Commodore Computer Club - Edizioni System Editoriale s.r.l. - v.le Famagosta, 75 - 20142 Milano - Tel. 02/8467348 - Dir. respons.: Michele di Pisa - Reg. Trib. di Milano n. 370/82 - Sped. Abb. post. gr. III/70 - Stampa: Lito 3 - Cologno Monzese.

Lire 7.000
Anno 2 - N. 8 - Gennaio 1986
Distr. MePe


I miei amici C 16 . Plus 4